# EXPERT SYSTEM
# IN
# SOFTWARE ENGINEERING
# USING
# STRUCTURED ANALYSIS AND
# DESIGN TECHNIQUE(SADT)

THESIS

Intaek Kim
Captain, ROKAF

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

1

AFIT/GCS/ENG/90J-02

# EXPERT SYSTEM IN SOFTWARE ENGINEERING USING STRUCTURED ANALYSIS AND DESIGN TECHNIQUE(SADT)

THESIS

Intaek Kim
Captain, ROKAF

AFIT/GCS/ENG/90J-02

Approved for public release; distribution unlimited

AFIT/GCS/ENG/90J-02

# EXPERT SYSTEM IN SOFTWARE ENGINEERING USING STRUCTURED ANALYSIS AND DESIGN TECHNIQUE(SADT)

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science (Computer Systems)

Intaek Kim, B.S.
Captain, ROKAF

June, 1990

Approved for public release; distribution unlimited

# *Acknowledgments*

This thesis effort consists of designing and implementing an Expert System which checks the syntax of the Structured Analysis and Design Technique (SADT) language from a drawing model generated during the requirements analysis phase of the software life cycle.

The development of the Expert System is separated into two parts. The first part is to translate a Structured Analysis diagram into a set of predicate data forms for SADT syntax analysis. The second part is to check the syntax of a Structured Analysis language through an inference engine and a rule-base knowledge.

I wish to express my gratitude to Dr. Gary B. Lamont, my thesis advisor, for his guidance and inspiration throughout this effort. Also, I wish to thank my committee members, Dr. Thomas C. Hartrum and Dr. Frank M. Brown, for their contribution to this thesis. I also would like to thank the R.O.K. and U.S. Governments for allowing me to have this experience.

In preparing this documentation, I thank the two gentlemen, Arthur L. Sumner and Terry L. Kitchen, for encouraging me and for English reviews. I would like to thank my father, my mother, and my wife, Jeunglim Jang, for their encouraging and support me. Finally, I want to thank my daughter, Yangheun.

To my father and mother.

Intaek Kim

## *Table of Contents*

# *List of Figures*

# *List of Tables*

Table Page

AFIT/GCS/ENG/90J-02

## Abstract

This thesis effort focuses on designing and implementing the Knowledge-Based Software Assistant System (KBSAS) for the Structured Analysis Design Technique (SADT) method developed by Softech, Inc.[1]

A Graphics Editor [2] is used to create specific Structured Analysis (SA) diagrams and a graphical symbol syntax is derived from these diagrams. The development of the KBSAS is divided into two parts: the design and implementation of a graphics translator and an application of a knowledge-based system for syntax checking.

First, the objective of the translator is to map a subset of the graphical symbol syntax from a SA diagram into the first order predicate calculus. The SA diagram information is represented in a set of predicate data forms.

Secondly, the objective of a knowledge-based system is to evaluate adherence to proper SADT syntax. This is accomplished by generating SA rules associated with either an activity box or boundary arrows. The requirements analyst and the designer are provided with a means of recovering from a graphical symbol syntax error(s) through a display window.

Specific emphasis focuses on a comprehensive mapping of the graphical symbol syntax to predicate logic as well as development of an application of a rule-based system using this capability.

[1] SADT is a trademark of Softech, Inc.

[2] developed by Steven E. Johnson at the Air Force Institute of Technology.

# EXPERT SYSTEM IN SOFTWARE ENGINEERING USING STRUCTURED ANALYSIS AND DESIGN TECHNIQUE(SADT)

## *I. Introduction*

### *Background*

There have been several thesis efforts in the field of Computer-Aided Software Engineering (CASE) tools to support the analysis and design stages of the software process at the Air Force Institute of Technology (AFIT). One of these efforts was a Graphics Editor for structured analysis with data dictionary support [1] (9).

The SAtool is one of several requirements analysis and design CASE tools based on the IDEF$_0$ syntax[2]. This tool provides a means of developing IDEF$_0$ diagrams [3] and data dictionary support. This tool also saves information derived from the diagrams; however, SAtool does not have the capability of checking IDEF$_0$ syntax. To solve this problem, a validation scheme for checking the consistency of IDEF$_0$ methodology and providing error messages with error recovery is required. Using the predicate data form, a specific IDEF$_0$ diagram from the SAtool is evaluated for adherence to proper SADT syntax through the use of a knowledge-based expert system (KBES). The earlier version of the validation tool was hosted on the SUN workstations[4] with the expert system written in Prolog-1 and run on a Zenith Z-248 computer (10).

[1] The Graphics Editor is called SAtool.

[2] IDEF$_0$ is a version of SofTech's SADT.

[3] Sometimes IDEF$_0$ diagrams are called SADT diagrams.

[4] SUN is a trademark of SUN Microsystems, Inc.

The intent of this thesis effort is to continue the earlier investigation by expanding the rule set, thereby developing a more integrated environment and analyzing its performance.

### *Statement of the Problem*

The specific objectives of this thesis investigation are to extend the earlier predicate calculus definitions of SADT syntax to the more complete set of SADT constructs, to extend the expert system rule set based on the new definitions, to integrate the graphical translation process in C with the expert system on a SUN workstation, and also if time permits, to use the structure of the knowledge-based SADT syntax system to incorporate the design knowledge of a specific software application.

### *Assumptions*

For the purpose of this investigation, several assumptions were made.

1. The primary users of this tool are AFIT graduate students and faculty.
2. The users of this tool are familiar with the Structured Analysis methodology.
3. The users of this tool are familiar with the SAtool.
4. Although not necessary, the users of this tool are also familiar with Prolog.

### *Research Approach*

The thesis objective is accomplished through the development of two major components: an IDEF$_0$ Diagram Translator and an IDEF$_0$ Syntax Expert System. The IDEF$_0$ syntax for SAtool was studied, followed by a review of the design and implementation of SAtool. The previous syntax rules of the syntax validation tool were also reviewed. These were updated and changed as necessary to reflect the development of the new system. The reusable components of the syntax validation tool were extracted, new code was written, and new syntactical rules were generated to implement the changes considered by the analysis.

The IDEF$_0$ Diagram Translator has three parts. The function of the first part is to create a set of predicate data forms from a SAtool activity box in the IDEF$_0$ Diagram in order to check the activity IDEF$_0$ syntax. The function of the second part is also to generate a set of the predicate data forms from the current IDEF$_0$ Diagram and its parent IDEF$_0$ diagram in order to check the boundary IDEF$_0$ syntax. In the third part, the objective is to create a file for the current IDEF$_0$ diagram in the form of a set of the predicate data to speedically check the boundary IDEF$_0$ syntax.

The IDEF$_0$ Syntax Expert System consists of two major parts: an inference engine and a rule base. The inference engine was selected as backward chaining strategy (search) called BC3[5] which is a directed problem-solving (pattern matching) process written in prolog. The rule base consists of activity rules and boundary rules. The activity rules check the 'activity' IDEF$_0$ syntax and the boundary rules are to check the 'boundary' IDEF$_0$ syntax.

The system checks IDEF$_0$ syntax, displays error messages, and provides editing suggestions interactively.

All software conformed to the software engineering standards in AFIT's *System Development Documentation Guidelines and Standards* draft #4 (8).

### *Materials and Equipment*

The materials and equipment for this effort were provided by the AFIT Department of Electrical and Computer Engineering. The following items were used:

1. SUN workstations.
2. Berkeley Unix [6] version 4.3.

[5] developed by Dr. Frank M. Brown at AFIT.

[6] Unix is a trademark of AT&T.

3. Suncore graphics and Suntools environment.

4. The software developed in this thesis effort.

5. Prolog environment.

*Overview of Thesis*

This thesis is divided into six chapters. Chapter I explains the history of AFIT's CASE tools based on SADT syntax and defines some of the terms to be used. Chapter II presents a literature review of the current issues that affect this thesis effort. The requirements for the translator and the expert system for this thesis effort are presented in chapter III. Chapter IV and V describe the high level and detailed design and implementation phases respectively. In chapter VI, the conclusions and the recommendations are addressed for this thesis effort.

# *II. Literature Review*

### *Introduction*

The focus of this thesis investigation is to design and implement an application of expert system formulation for checking the syntax of IDEF$_0$ diagrams as derived from SAtool. The current SAtool is one of requirements analysis CASE tools based on the IDEF$_0$ syntax which is a subset of SADT syntax. The purpose of this literature review is to discuss the issues of the requirements analysis phase, AFIT Structured Analysis Syntax as a subset of IDEF$_0$ syntax, and a rule-based expert system architecture.

### *Requirements Analysis Phase*

The software life cycle represents the functionally distinct portions of development and use of a software product from birth to death. The classic life cycle model as shown in Figure 2.1 may be divided into six major phases: system engineering and analysis phase, software requirements analysis phase, design phase, implementation, testing, and finally maintenance phase (3). The requirements analysis process focuses specifically on software by definition. To understand the nature of the software to be built, the software analysts must understand the information domain for the software, as well as the required function, performance expectations, and interfacing (12). Analysts should develop the software specification using a documentation tool so that they may later compare the requirements against the solution because a complete specification of software requirements is imperative to the success of a software development effort. No matter how well-designed or well-coded, poorly specified software will disappoint the user and bring grief to the developer (12). A number of software analysis and specification methods have been developed and each method has its own notation and point of view; however, there is a set of general principles for requirements analysis:

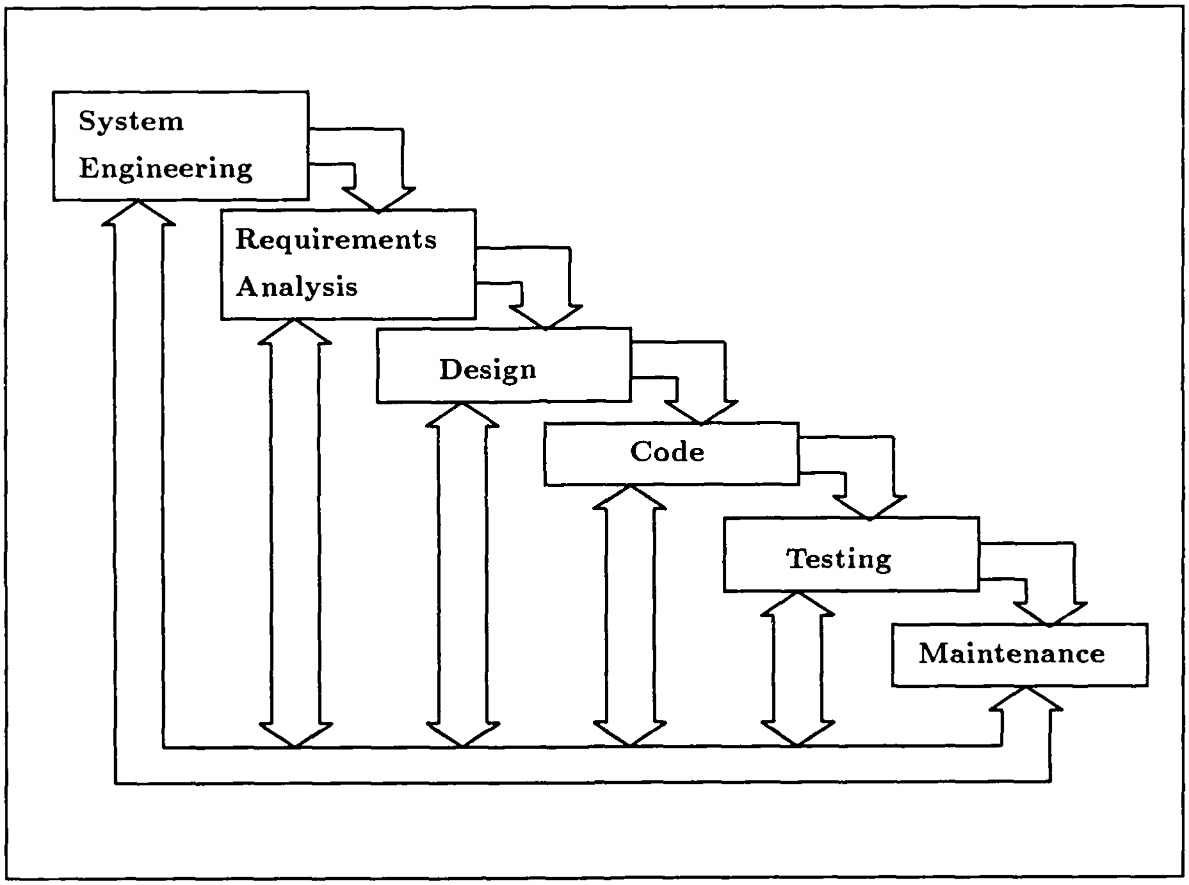


Figure 2.1. Classic Software Life Cycle Model(12:20)

1. The information domain and the functional domain of a problem must be represented by syntax and understood by humans.
2. The problem must be partitioned in a manner that uncovers detail in a hierarchical fashion.
3. Logical and physical representations of the system should be developed (12).

Many requirements analysis methods and tools have been developed during the past decade. The methods and tools may be divided into three broad analysis categories: data flow-oriented analysis, data structure-oriented analysis and language-based formal specification (12). The software requirements analysis methods were originally developed to be applied manually; however, each of these methods is available in a computer-aided format (12). Several of computer-aided analysis tools have been developed to automate the generation and maintenance of what was originally a manual method. These tools make use of a graphical notation for analysis. This class of tools produces diagrams, aids in problem partitioning and maintains a hierarchy of information about the system (12). These CASE tools enable the analyst to update information and compare the connections between new and existing representations of the system. For example, the SAtool enables the analyst to produce a structured analysis diagram and a data dictionary and maintain these in a data base that can be analyzed for correctness, consistency, and completeness. The computer-aided requirements analysis approach provides benefits as followings:

- improved documentation quality through standardization and reporting
- better coordination among analysts in that the data base is available to all
- gaps, omissions, and inconsistency are more easily uncovered through cross-reference maps and reports
- the impact of modifications can be more easily traced
- maintenance costs for the specification are reduced (12:200).

*AFIT Structured Analysis Syntax*

The SADT syntax is based on a tabulation of some 40 notations in a paper by Douglas T. Ross of Softech, Inc. (14). The notations give the definitions and the semantics of the SADT graphic language. The SADT methodology provides a means of handling large complex system problems. The SADT notations consist of two major constructs: rectangular boxes and arrows. Rectangular boxes, identified as verbs (activities), provide for the decomposition of the system parts. Arrows, labeled with nouns (data structures), represent the data flow relationship among the rectangular boxes. The rectangular boxes, arrows and English text build a diagram which represents the whole system.

The U.S. Air Force Program for Integrated Computer-Aided Manufacturing (ICAM) has developed the IDEF$_0$ (ICAM Definition Method Zero) [1] language. IDEF$_0$ syntax is a derivative of the SADT syntax and is used for software requirements analysis. The AFIT Structured Analysis syntax implemented by SAtool is represented in Table 2.1 (9).

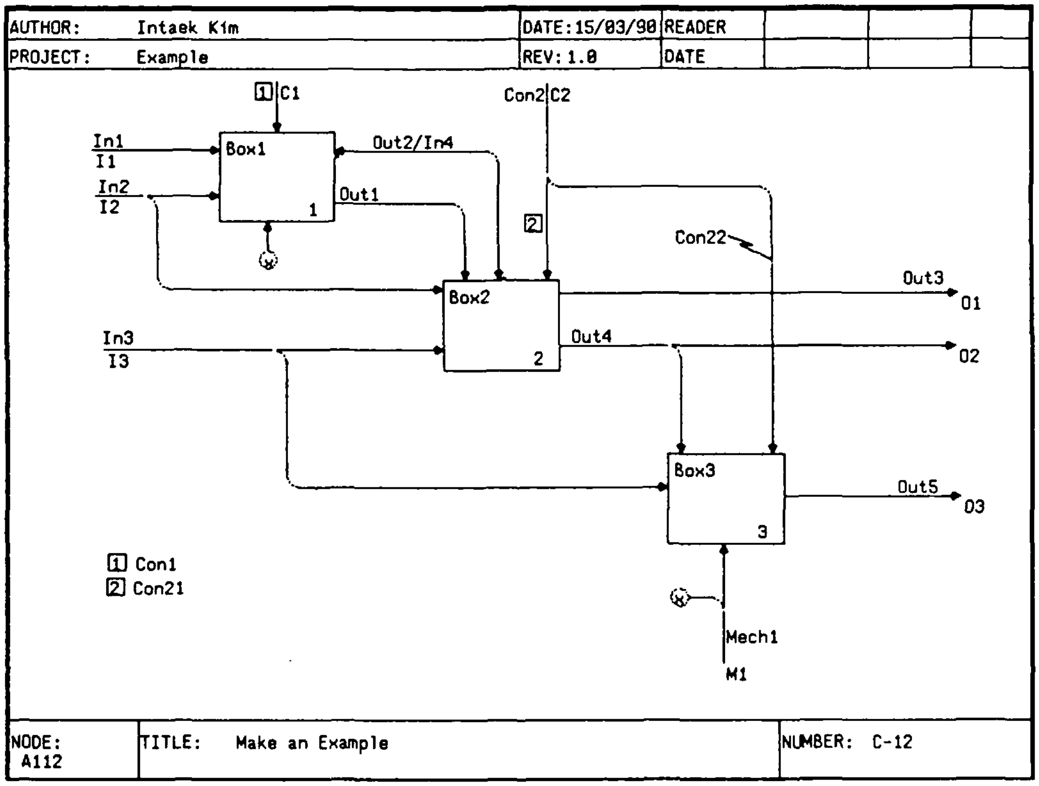
Column 1 in the table shows the line numbers of the SADT graphical notations (14:20). Column 2 shows the name by which each notation was referenced in SAtool. Column 3 indicates the page in the SAtool User's Manual (9).

The SAtool provides a means of drawing the IDEF$_0$ diagrams and storing information derived from the diagrams. Each diagram is drawn and stored individually. An example of an IDEF$_0$ diagram drawn by SAtool is shown in Figure 2.2. *Box1, Box2,* and *Box3* are for ACTIVITY NAMEs and the numbers in the rectangular boxes represent NODE NUMBERs. The numbers in small rectangular boxes show FOOTNOTEs. The label in a small circle is for TO/FROM ALL. *In1, Out2, Con22, Mech1*, and etc. represent line LABELs for INPUT, OUTPUT, MECHANISM, and

[1] See the reference 17. ICAM Definition method IDEF$_0$ Sep.1979.

| Ross Article Line Number | Term | User's Manual Reference |
|---|---|---|
| 1 | BOX | 2-2,3 |
| 2 | ARROW | 2-2,3 |
| 3 | INPUT | 3-26 (FIG) |
| 3 | OUTPUT | 3-26 (FIG) |
| 4 | CONTROL | 3-26 (FIG) |
| 5 | MECHANISM | 3-11 |
| 6 | ACTIVITY NAME | 2-3,4 |
| 7 | LABEL | 2-3,4 |
| 12 | BRANCH | 3-9 |
| 13 | JOIN | 3-9 |
| 14 | BUNDLE | 6-14 |
| 15 | SPREAD | 6-14 |
| 18 | BOUNDARY ARROW | 3-17 |
| 20 | DETAILED REFERENCE NUMBER | 2-3 |
| 22 | 2-WAY ARROW | 3-26 (FIG) |
| 24 | TUNNEL ARROW | 2-3 |
| 25 | TO/FROM ALL | 6-21 |
| 27 | FOOTNOTE | 3-26 (FIG) |
| 28 | META-NOTE | 2-3 |
| 29 | SQUIGGLE | 3-26 (FIG) |
| 30 | C-NUMBER | 2-3 |
| 31 | NODE NUMBER | 2-3 |
| 32 | MODEL NAME | 3-26 (FIG) |
| 33 | ICOM CODE | 4-8 |
| 37 | FACING PAGE TEXT | 4-1 |
| 38 | FEO (FOR EXPOSITION ONLY) | 6-5 |
| 39 | GLOSSARY | 2-3 |
| 40 | NODE INDEX | 2-3 |

Table 2.1. AFIT SADT syntax used by SAtool (9:A-3)



|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
| AUTHOR: | Intaek Kim | DATE:15/03/90 | READER |  |  |  |
| PROJECT: | Example | REV: 1.0 | DATE |  |  |  |

|  |  |  |
|---|---|---|
| NODE:<br>A112 | TITLE: Make an Example | NUMBER: C-12 |

Figure 2.2. An example of IDEF$_0$ diagram

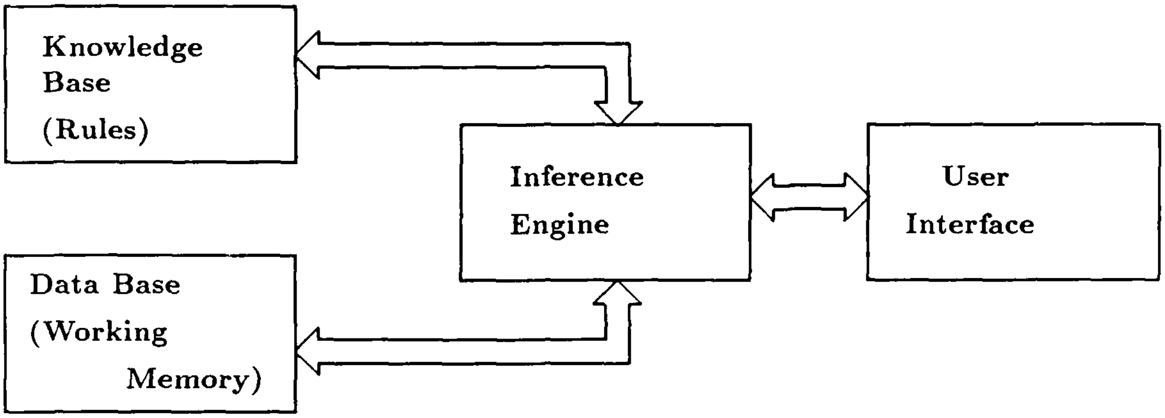


Figure 2.3. A typical Expert System Structure

CONTROL. *I1, C1, O1, M1*, and etc. also represent ICOM CODEs for boundary arrows.

*Expert System*

> *Knowledge-based expert systems are likely to be applied to requirements analysis tasks. However, the definition of the knowledge base (facts, rules, and necessary inferences to perform analysis) will remain a significant challenge in the foreseeable future.* (12:201)

Figure 2.3 presents the components of a general expert system: knowledge base (rules), data base (facts; working memory), inference engine, and user interface.

The knowledge base can be represented by many different methods, such as predicate calculus, lists, frames, semantic nets, production rules, etc. In this thesis effort, the language of if-then rules was selected to represent the knowledge base, since it provides several features which are modularity, incrementability, modifiability, and transparency of the system. The if-then rule consists of two parts : condition and conclusion. The logical condition part may contain one or more premises linked by the conjunctions AND, OR, or NOT. If the conditions are true (met), the conclusion part is also true (fired).

The data base is a portion of working memory where the current status of problem is stored. Initially, the lists of object, attribute, and value (OAV) triple derived from a IDEF$_0$ diagram are stored. Then the new lists of OAV triple from the inference process are added. The data base also stores a list of rules that have been examined, and fired in some order. The rule order can be given later if the user or developer requires an explanation of the reasoning process.

The inference engine is called a rule interpreter because its operation is like a software interpreter for a computer language. The rule interpreter examines the rules in a specific order searching for matches to the initial and current conditions given in the working memory. As the rules continue to fire, they will reference one another and form an inference chain. The firing of a rule may add new facts to the working memory, which gives the rule interpreter additional information on which to proceed. This process continues until the solution is found.

Given a desired goal, there are two basic approaches in searching for a solution: forward chaining and backward chaining. In *forward chaining*, the rule interpreter tries to match a fact in the working memory to the situation stated in the condition part. If a fact in the working memory has been matched, then that rule is fired. The conclusion part could generate a new fact that is stored in the knowledge base. This new fact may be used in the search for the next proper rule. This process continues until the solution of the given goal is satisfied. In *backward chaining*, the rule interpreter starts examining the conclusion part to look for a match. If a match is found, then the working memory is updated recording the conditions that the rule stated as necessary for supporting the matched conclusion (13). The backward chaining process continues with the system repeatedly attempting to match the conclusion part against the current system's status. The corresponding condition parts matched are used to produce new intermediate goal states which are recorded in the working memory. This process continues until the given goal is proved.

Finally, the user interface provides a means of communication between the user and the system. The user interface asks questions or presents menu choices for entering initial facts in the data base. Any intermediate communications during the problem-solving process are taken care of by the user interface.

Expert systems are far more useful if they have the following additional features:

- *Modularity:*

  Each rule defines a small, relatively independent piece of knowledge.

- *Incrementability:*

  New rules can be added to tl. ; knowledge base relatively independently of other rules.

- *Modifiability (as a consequence of modularity):*

  Old rules can be changed relatively independently of other rules.

- *Support system's transparency* (4:316-317).

### *Summary*

This thesis effort concentrates on translating an IDEF$_0$ diagram drawn by the SAtool into a set of predicate data forms during requirements analysis. It also focuses upon developing an application of a rule-based expert system for evaluating IDEF$_0$ syntax. The literature review in this chapter provides understanding concerning the main subjects of this thesis investigation: requirements analysis phase of software development, IDEF$_0$ syntax, SAtool, and rule-based expert system components.

# *III. System Requirements Analysis*

## *Introduction*

This thesis investigation is classified into two major categories. First, the *IDEF$_0$ Diagram Translator* is to be redesigned and reimplemented to translate any IDEF$_0$ diagrams drawn by SAtool into a set of predicate data forms. It should create a necessary file to be used for checking IDEF$_0$ syntax. The second category is to design and implement the *IDEF$_0$ Syntax Expert System* which is an application of a knowledge-based expert system.

This chapter presents the considerations related to the development of the IDEF$_0$ Diagram Translator requirements, IDEF$_0$ Syntax Expert System requirements, formalization criteria, the IDEF$_0$ diagrams of this tool, and validation test requirements.

## *Considerations of the Previous Studies*

The SAtool provides an interactive graphics editor for drawing IDEF$_0$ diagram and a means of generating data dictionary information (9:3-2). The SAtool also provides the capability to check IDEF$_0$ syntax for an activity box in any IDEF$_0$ diagrams(10:2-1). The SAtool does not provide a means of checking IDEF$_0$ syntax for any boundary arrows with the parent IDEF$_0$ diagram, however, the SAtool provides a means of checking IDEF$_0$ syntax for only one activity box in any IDEF$_0$ diagram. The current SAtool checks IDEF$_0$ syntax through Zenith Z-248 workstations using the expert system written in Prolog-1. The improved tool in this thesis effort should interface with the SAtool, produce a set of predicate data forms for the activity boxes and the boundary arrows information in any IDEF$_0$ diagrams, and provide the more completed capability of checking IDEF$_0$ syntax. Also, the revised tool should satisfy all requirements of the previous SAtool (9).

### *IDEF$_0$ Diagram Translator Requirements*

The function of the IDEF$_0$ Diagram Translator is to generate the predicate data forms from any IDEF$_0$ diagrams. The IDEF$_0$ Diagram Translator should act as a bridge between the current SAtool and the IDEF$_0$ Syntax Expert System. The current SAtool was written in the C language and used graphics software packages called SunView and SunWindow environment. Thus, the IDEF$_0$ Diagram Translator should operate with the current SAtool. Since the current SAtool provides a means of checking IDEF$_0$ syntax for only one activity box, to check syntax for boundary arrows, the IDEF$_0$ Diagram Translator should be redesigned and reimplemented in the C language. The predicate data forms generated by the IDEF$_0$ Diagram Translator should be the initial data base for the IDEF$_0$ Syntax Expert System. It is necessary to formalize the IDEF$_0$ syntax to produce the predicate data forms and to check the IDEF$_0$ syntax in the IDEF$_0$ diagrams. IDEF$_0$ syntax is formalized as follows:

1. The formal definition of IDEF$_0$ syntax must contain the syntax information in any IDEF$_0$ diagram and be described syntactically,
2. provide the means to determine syntax errors in any IDEF$_0$ diagram,
3. provide a domain where the definition of consistency can be given,
4. serve as the final arbiter in cases where there is disagreement concerning the exact meaning of the representation, and
5. should be able to be implemented in a computer system (10:2-3).

The next section, *Requirements analysis Diagrams*, describes the functional decompositions for the IDEF$_0$ Diagram Translator.

*IDEF$_0$ Syntax Expert System Requirements*

The IDEF$_0$ Syntax Expert System should allow the user to check the activity IDEF$_0$ syntax and the boundary IDEF$_0$ syntax in any IDEF$_0$ diagrams using a created predicate data file.

The backward chaining search is useful when there are many more rules than desired goals. A backward chaining inference engine was selected called BC3[1] which directed problem-solving processes and acts as a rule interpreter (6) because the backward chaining strategy is useful when there are many more rules than desired goals. The rule base should be able to support the knowledge of IDEF$_0$ syntax with the inference engine in accordance with the activity boxes and the boundary arrows. To simplify the rule base, the rule base should be consisted of two separate parts because the boundary arrows information is not necessary in checking the activity IDEF$_0$ syntax and the activity boxes information is not needed for examining the boundary IDEF$_0$ syntax. The first part, called the Activity IDEF$_0$ Syntax rule base, should allow the checking of the Activity IDEF$_0$ Syntax using a created predicate file which includes all information pertaining to an activity box in any IDEF$_0$ diagram. The second part, called the Boundary IDEF$_0$ syntax rule base, should allow for the checking of the Boundary IDEF$_0$ Syntax using a created predicate file which includes all information pertaining to the boundary arrows in any IDEF$_0$ diagram and its parent IDEF$_0$ diagram.

*Validation Test Requirements*

Some parameters can be used for evaluating the conformity of the requirements with the tool. As mentioned in Chapter I, the important parameters are the accuracy with which the tool checks the IDEF$_0$ syntax and the capability of which the tool interactively displays error messages and editing suggestions. Other parameters to be considered are user friendliness, maintainability, compatibility, and consistency.

[1] developed by Dr. Frank M. Brown at AFIT.

*Requirements Analysis Diagrams*

This section presents the functional model which defines and describes the functional decompositions for *the* IDEF$_0$ *Diagram Translator and the* IDEF$_0$ *Syntax* Expert System discussed in the previous sections. The IDEF$_0$ diagrams in this section are based on the analysis of processes or *activities* and illustrate one level of the functional decomposition with the facing page text. The facing page text provides additional information that is not easily inferred from the diagram. The Data Dictionary entries provide even more detailed information in accordance with each activity and data item (22:4-5).

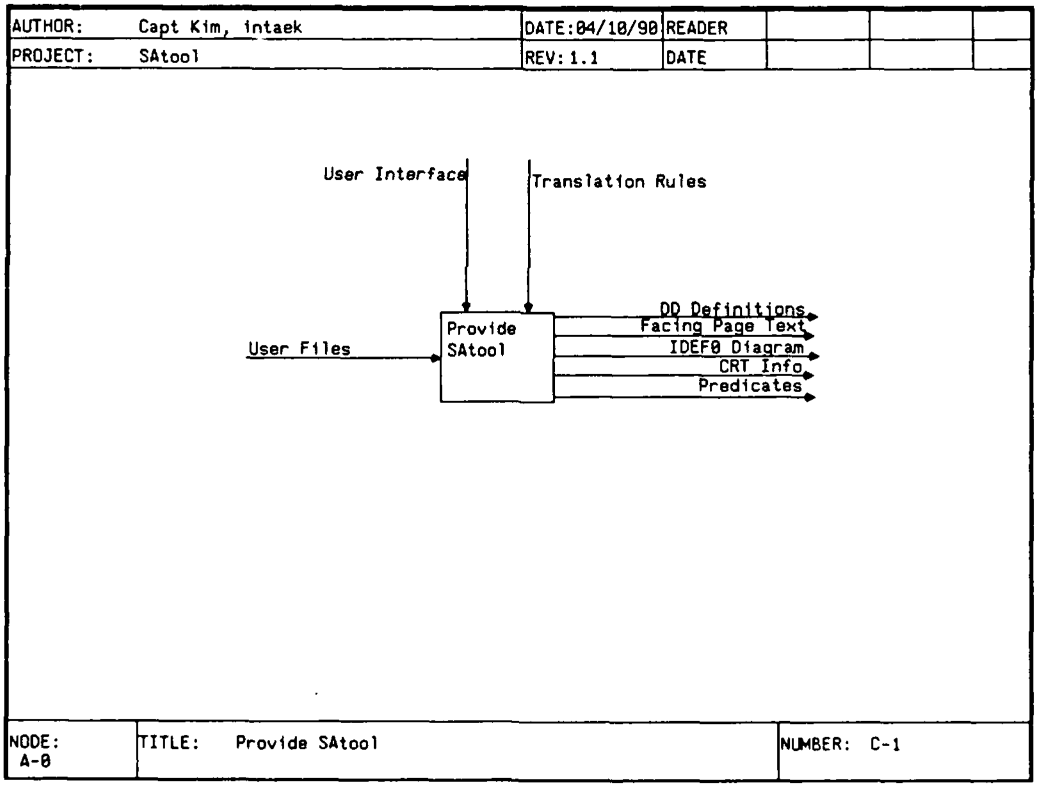
Figure 3.1 shows the top most level IDEF$_0$ diagram for the overall system of the SAtool. The purpose of the *Provide SAtool* function is to create and edit an IDEF$_0$ diagram, its data dictionary information, and the facing page text interactively (9). The function also involves the process which produces the predicate data forms to be used for the knowledge base of the IDEF$_0$ Syntax Expert System.

Figure 3.2 displays the first decomposition of the top most level of *Provide SAtool* activity. This decomposition shows the two primary functions: *Provide SA Editor* and *Translate Diagram.* The *Provide SA Editor* function is to draw the activity IDEF$_0$ diagram and to generate its data dictionary information and its pacing page text for activity and data entry. The *Translate Diagram* activity provides a means of translating IDEF$_0$ diagrams into predicate data forms.

Figure 3.3 shows the decomposition of the *Translate Diagram* activity into three functional components:*Translate Activity*, *Translate Boundary*, and *Save Diagram.* The *Translate Activity* provides a means of translating an activity box in any IDEF$_0$ diagrams into a set of predicate data forms through the translation rules of the activity box and saving it into a temporary file. The *Translate Boundary* operation is the process which translates the boundary arrows in any IDEF$_0$ diagrams into a set of predicate data forms through the translation rules of the boundary arrows and saves it into a temporary file. Finally, *Save Diagram* provides a means of translating

A-0 Provide SAtool

Abstract: Provide SAtool provides a means of mechanism by which the user is able to draw Activity IDEF0 Diagrams. From these diagrams, Facing Page Text and Data Dictionaries for Activities and Data and Predicate Data forms are generated.



|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
| AUTHOR: | Capt Kim, intaek | DATE:04/10/90 | READER |  |  |  |
| PROJECT: | SAtool | REV: 1.1 | DATE |  |  |  |

|  |  |  |
|---|---|---|
| NODE:<br>A-0 | TITLE: Provide SAtool | NUMBER: C-1 |

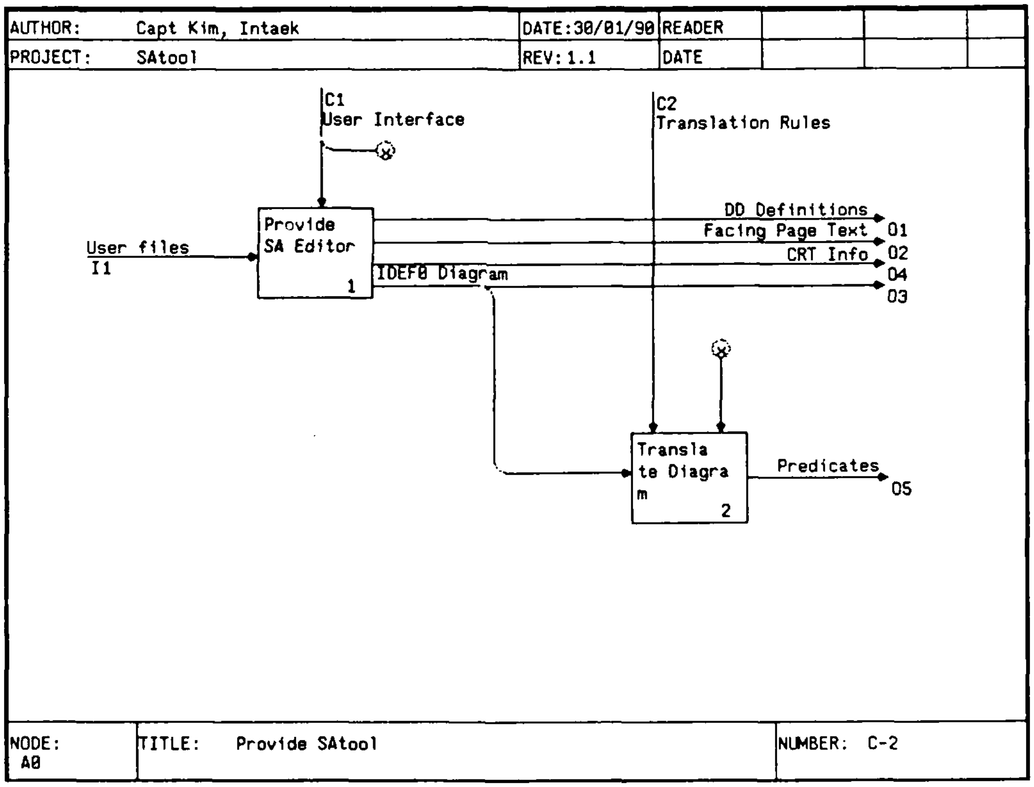
Figure 3.1. Provide SAtool

A0 Provide SAtool

Abstract: Provide SAtool provides the user a mechanism by which the user is able to draw Activity IDEF0 Diagrams. From these diagrams, Facing Page Text, Data Dictionaries for Activities and Data, and Predicate Data forms are generated.

A1 Provide SA Editor provides a means of drawing Activity IDEF0 Diagrams. From these diagrams, Facing Page Text and Data Dictionaries for Activities and Data are generated.

A2 Translate Diagram provides a means of translating IDEF0 Diagrams into Predicate Data Forms.



| | | | | | | |
|---|---|---|---|---|---|---|
| AUTHOR: | Capt Kim, Intaek | DATE:30/01/90 | READER | | | |
| PROJECT: | SAtool | REV: 1.1 | DATE | | | |

NODE: A0 | TITLE: Provide SAtool | NUMBER: C-2

Figure 3.2. Provide SA Editor

the IDEF$_0$ diagram into a set of predicate data forms and saving it into a file which is specified by the user. Further functional decompositions are presented in Appendix A.

### *Summary*

This chapter presented the requirements analysis for the development of IDEF$_0$ Diagram Translator and IDEF$_0$ Syntax Expert System. Since this investigation extends the earlier version of the SAtool, this tool should satisfy all requirements of the earlier version. This tool should also provide a means for displaying error messages and editing suggestions. The functional decompositions for this tool was presented in *Requirements Analysis Diagrams* section.

The next two chapters use the requirements developed in this chapter as the fundamental for designing, implementing, and testing of IDEF$_0$ Diagram Translator and IDEF$_0$ Syntax Expert System.

## A2 Translate Diagram

Abstract: Translate Diagram provides the user a means of translating IDEF0 Diagrams into Predicate Data Forms.

A21 Translate Activity provides a means of translating an activity box in any IDEF0 Diagrams into a set of predicate data forms through the translation rules of the activity box.

A22 Translate Boundary provides a means of translating the boundary arrows in any IDEF0 Diagrams into Predicate data forms through the translation rules of the boundary arrows.

A23 Save Diagram provides a means of translating the IDEF0 Diagram into a set of the predicate data forms and saving it into a file which user specifies interactively.

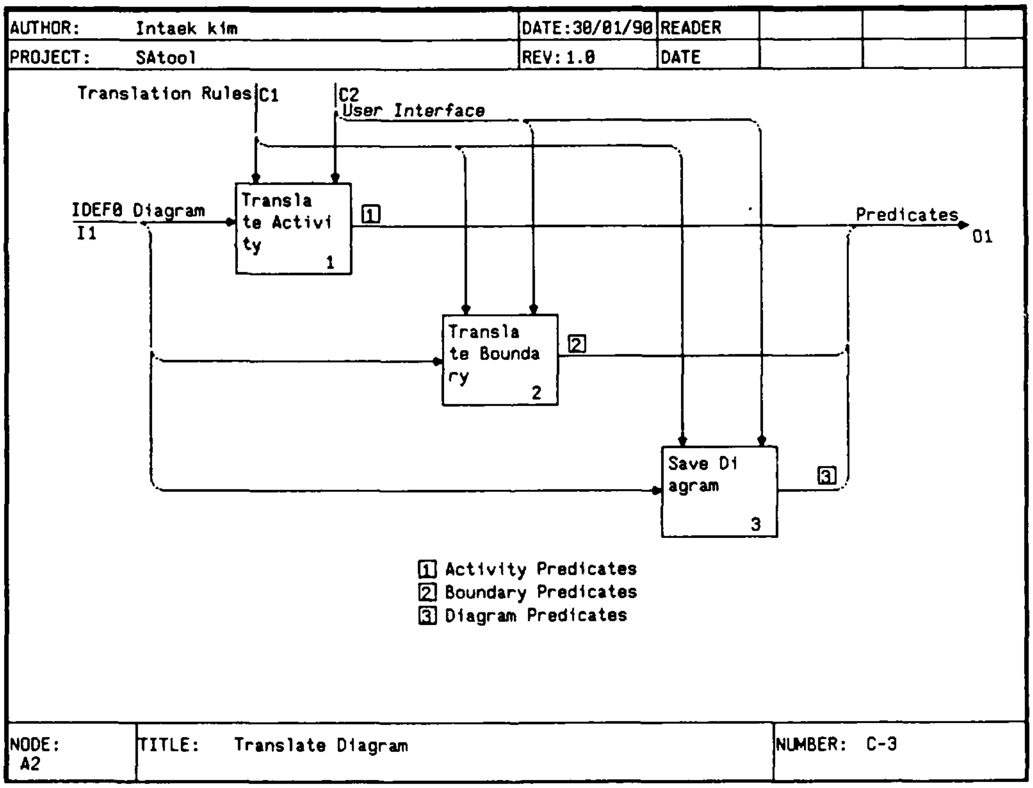


Figure 3.3. Translate Diagram

# *IV. High Level Design*

## *Introduction*

The purpose of this chapter is to present and justify the preliminary software design for the IDEF$_0$ Diagram Translator (IDT) and the IDEF$_0$ Syntax Expert System (ISES). Throughout the remainder of this investigation, IDT and ISES refer to these two particular systems. Preliminary design is associated with the transformation of requirements into the software architecture. The transformation starts with several considerations of previous studies addressed in Chapter III. The modified screen layout and menu selection are then presented. In addition, the main functions of the IDEF$_0$ Diagram Translator, the IDEF$_0$ Syntax Expert System, and the associated components are introduced.

## *Previous Study Considerations*

Since the tool should interface with the SAtool, the hardware and the software to be used are already chosen. Thus, the Sun3 and the Sun4 workstations using the SunOS and the SunView window-based environment are ıequired for this tool because the SAtool was developed through the Sun workstation. Also since the IDEF$_0$ validation tool was implemented with the C language in order to translate the IDEF$_0$ diagram into the predicate data forms, the C language is used to expand the capability of the translating process. This decision is reasonable because many portions of the validation tool and the SAtool could be directly reused. Appendix C represents a summary of the data structures which are used for the earlier version of SAtool and this thesis effort.

## *Screen Layout and Menu System*

In this thesis investigation, the screen layout of the SAtool and the validation tool should be modified by adding new menu items for checking IDEF$_0$ syntax.

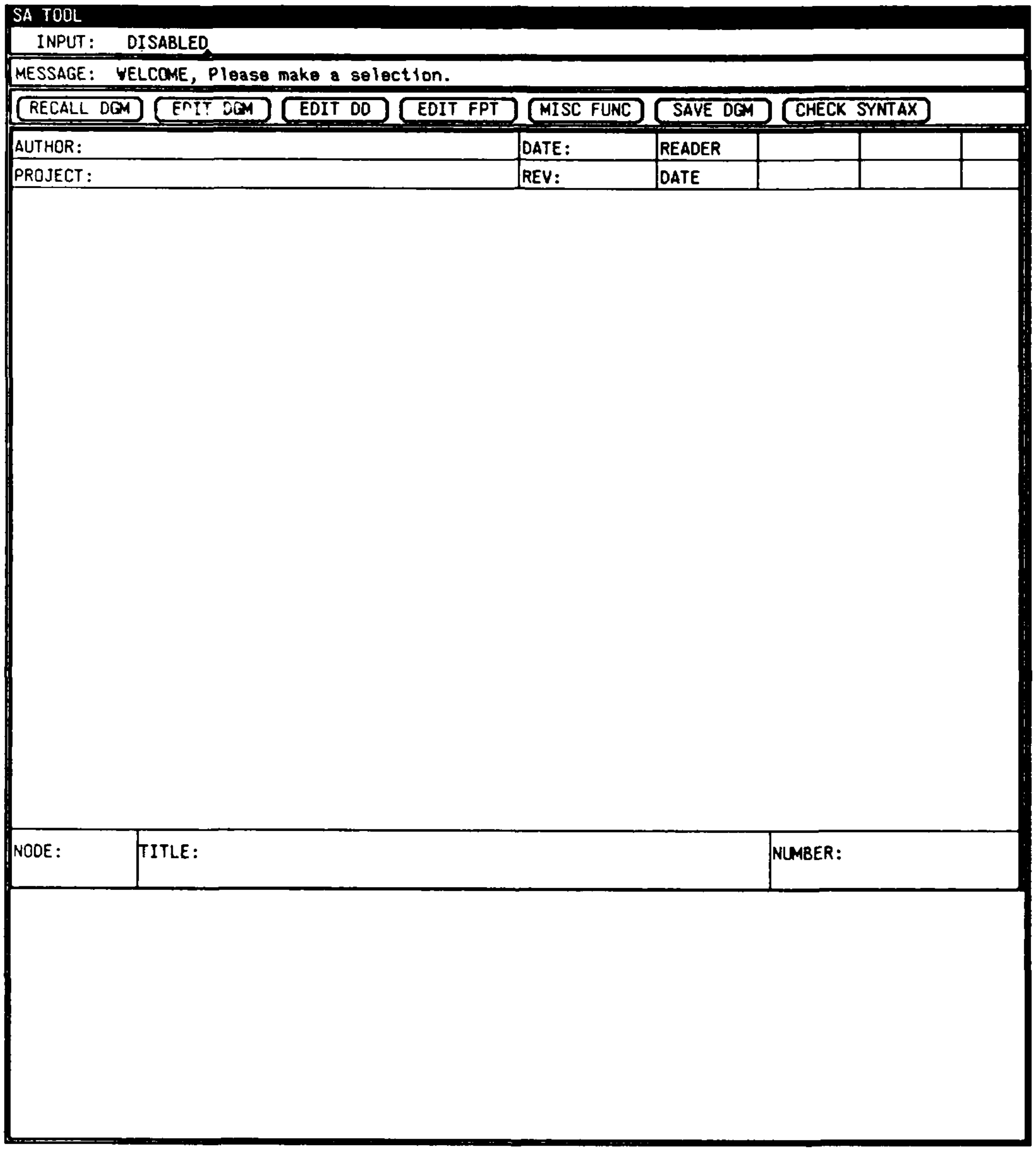
Figure 4.1 is a picture of the actual screen layout used by the tool.

The actual screen is divided into five windows: the Input Window, the Message Window, the Selection Window, the Diagram Window, and the Data Dictionary Window in vertical order. The function of each window is the same as in the SAtool except the function of the Selection Window. The Selection Window, located directly below the Message Window is used for selecting the menu which contains the next desired operation. The six ovals arranged in left to right order are: *RECALL DGM, EDIT DD, EDIT FPT, MISC FUNC, SAVE DGM,* and *CHECK SYNTAX.* The function of *RECALL DGM* is to read a previously saved IDEF$_0$ diagram. The *EDIT DGM* function is to create and edit an IDEF$_0$ diagram according to its attribute menus. The function of *EDIT DD* is to create and edit data dictionary information. The function of *EDIT FPT* is to edit facing page text of an IDEF$_0$ diagram. The function of *MISC FUNC* is to save an IDEF$_0$ diagram, change directory, start new diagram, and exit SAtool. The function of *SAVE DGM* is to save the graphics information (.gph extension) and the data dictionary information (.dbs extension) in files under the name provided by the user. Finally, The *CHECK SYNTAX* function is to translate and save IDEF$_0$ diagrams as predicate data forms and to check IDEF$_0$ syntax. Figure 4.2 presents a picture of all menu selections to be used and their decompositions.

Since the functions of all menu selections are the same as those of the SAtool except *Activity, Boundary,* and *Save (.pro)* selections, the other detailed descriptions of the functions of the menu selections except for above three are available in reference (9). Further descriptions of *Activity, Boundary,* and *Save (.pro)* selection functions are discussed in the next section.

### *IDEF$_0$ Diagram Translator*

The translator for the IDEF$_0$ diagram is used to translate the IDEF$_0$ graphical features into predicate data forms. The requirements for the formalization criteria



INPUT: DISABLED

MESSAGE: WELCOME, Please make a selection.

| | | | | | |
|---|---|---|---|---|---|
| AUTHOR: | DATE: | READER | | | |
| PROJECT: | REV: | DATE | | | |

| | | |
|---|---|---|
| NODE: | TITLE: | NUMBER: |

Figure 4.1. Screen Layout

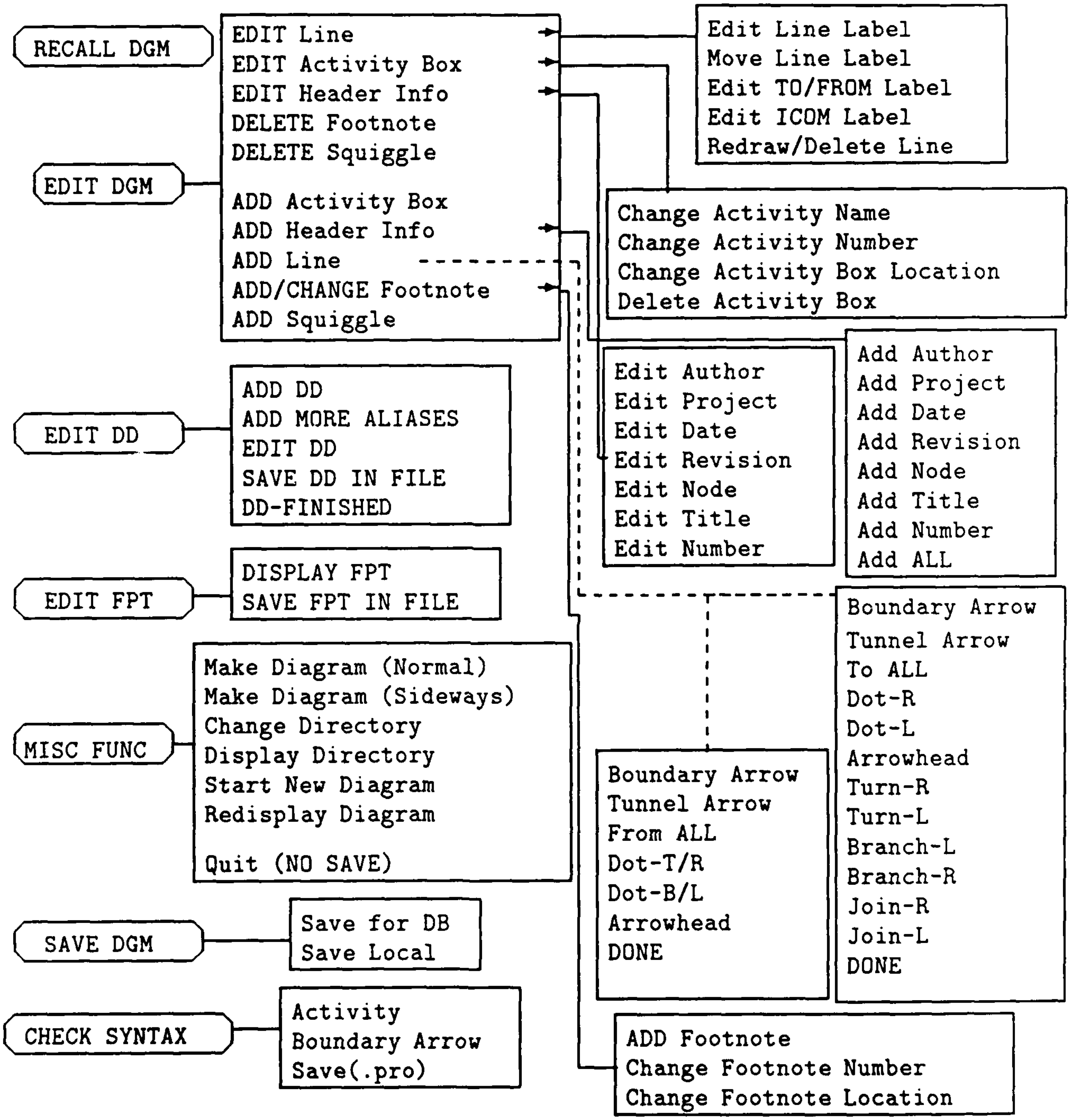


Figure 4.2. SAtool Menus

have been discussed in Chapter III. There are many ways to formalize the graphical language such as PSL/PSA, TAGS, SREM, and etc (12:163-209). Also, there are lots of different ways to represent knowledge for expert system, for example rules, semantic nets, frames, scripts, predicate logic, and others (13:63-102). In this thesis investigation, predicate logic is used to translate an IDEF$_0$ diagram into a set of predicate data forms. The predicate logic is often used as a means of knowledge representation in expert systems and is the basis for logic programming. Many specialists r^gard it as the single most important knowledge representation method. The predicate logic also provides for symbols to represent objects and then allows these symbols to be used as components of statements. As shown in Figure 4.3, the IDEF$_0$ graphical notations used by the SAtool consist of two major constructs: activity box and arrow. The informations related to an activity box are *ACTIVITY NAME, ACTIVITY NUMBER, INPUT, OUTPUT, CONTROL,* and *MECHANISM.* Although there are many types of arrows such as *BRANCH, JOIN, SPREAD, BUNDLE,* and others, the arrow type can be considered to be one of either the boundary arrow type connected on the parent IDEF$_0$ diagram, the tunnel arrow type, or the interboxes arrow type which is connected between two boxes in the same IDEF$_0$ diagram. Among the arrow types, only the boundary arrow type has the information of the relationships between an IDEF$_0$ diagram and its parent IDEF$_0$ diagram. This boundary arrow type can be considered as either boundary *INPUT, OUTPUT, CONTROL,* or *MECHANISM.* The other arrow types are related to an activity box in an IDEF$_0$ diagram. Thus, it is useful to translate and create separately the graphical informations for an activity box and the boundary arrows. Another reason for translating the IDEF$_0$ diagram into a set of predicate data forms and generating seperately its information is to reduce the size of the predicate data file for checking IDEF$_0$ syntax and to provide simplicity of the knowledge base as well. Therefore, the function of the IDEF$_0$ Diagram Translator is divided into three parts. The function menus of CHECK SYNTAX in Figure 4.2 show the functions of the IDEF$_0$ Diagram Transla-

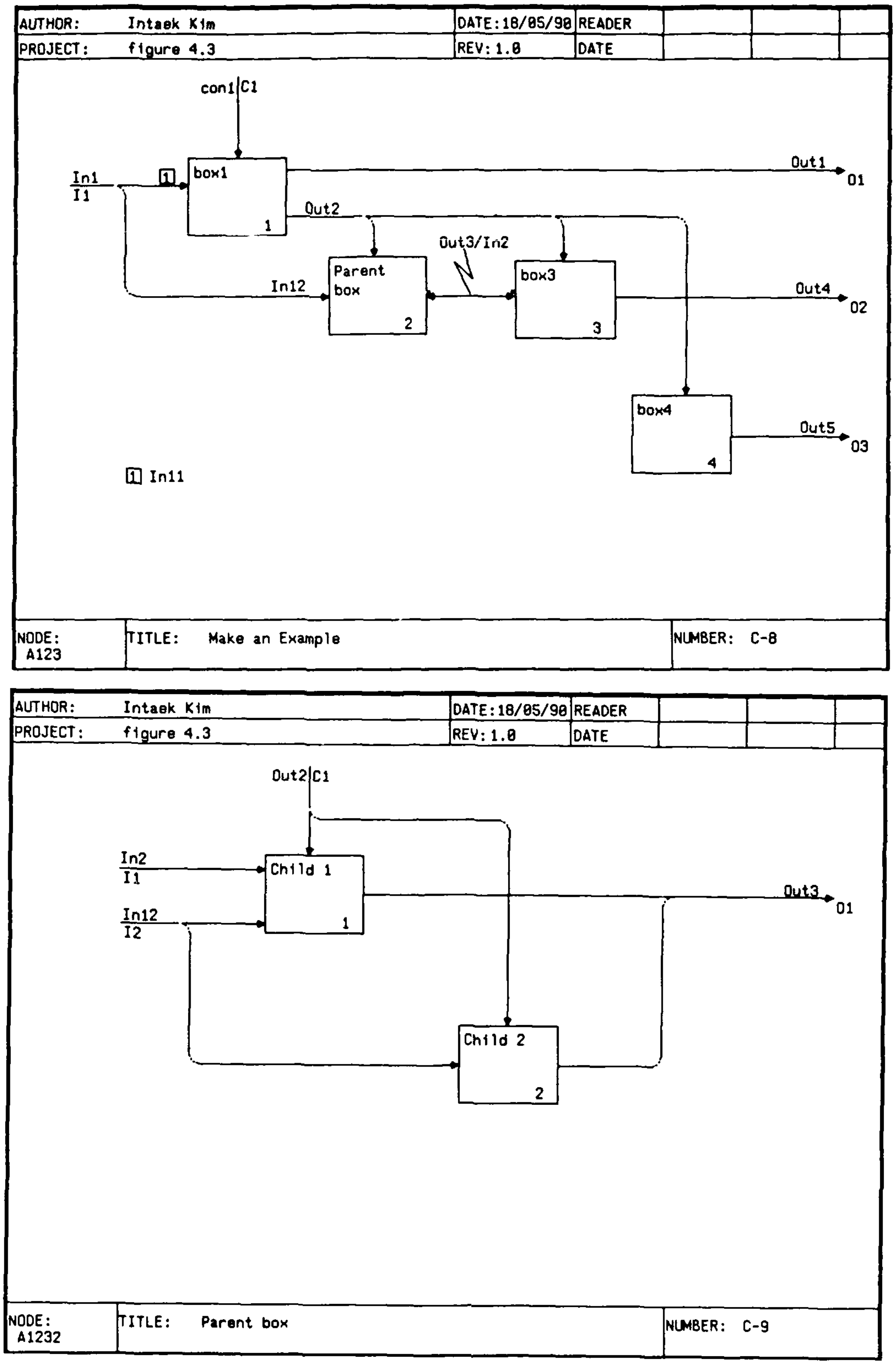


Figure 4.3. IDEF$_0$ Diagram with Parent

tor. The *Activity* selection menu is used to translate and create a file of the predicate data forms automatically from an activity box clicked on by the user in any IDEF$_0$ diagrams. The *Boundary Arrow* selection menu is used to translate and create automatically a file of the predicate data forms from the boundary arrow information in the current IDEF$_0$ diagram and the predicate data file of its parent IDEF$_0$ diagram. Finally, *Save (.pro)* selection menu is used to translate an IDEF$_0$ diagram into a set of predicate data forms and to save it to a file that the user specifies. This saved file is to be used for checking boundary IDEF$_0$ syntax with the next decomposed IDEF$_0$ diagram

### *IDEF$_0$ Syntax Expert System Components*

The IDEF$_0$ Syntax Expert System provides a means of checking the IDEF$_0$ syntax in any IDEF$_0$ diagrams using a created predicate data file. The components of ISES are the inference engine (control strategy) which is selected as BC3, the knowledge base (rule base) which consists of the IDEF$_0$ syntax rules, the data base (working memory) which is made up of a list of facts derived from the IDEF$_0$ diagram, and the user interface which supplies facts or other information to the ISES and transmits expert advise to the user as shown in Figure 2.3.

*Inference Engine.* An inference engine applies the knowledge to the solution of a specific problem. In general, the same control strategy can be used to reason out all kinds of actual problems because it is separated from the knowledge base for the particular application. One of the inference rules, called modus ponens, is used in producing a proof which allows a fact or truth to be inferred from two other statements. For example, if propositions $P$ and $P$ *implies* $Q$ are true, then proposition Q is true. The inference engine repeatedly applies the method of modus ponens to the knowledge base to extract a specific value or solve a particular problem. During consultation of the IDEF$_0$ Syntax Expert System, the following functions should be performed:

- interface with users
- obtain and load the knowledge base
- apply the rules in the knowledge base and the facts in the working memory to achieve goals
- display the messages about the results of checking $\text{IDEF}_0$ syntax.

In Chapter II, the backward chaining control structure has been represented. Backward chaining is often beneficial when there are many more facts than final goals (thus called goal-driven reasoning). Since the number of facts in any $\text{IDEF}_0$ diagram is many more than that of goals for $\text{IDEF}_0$ Syntax, backward chaining is useful for the ISES design.

*Knowledge Base.* The heart of the expert system is the knowledge base, which contains the problem-solving knowledge of the particular application. In the $\text{IDEF}_0$ Syntax Expert System, the knowledge base is represented in the form of *if...then* rules and is separated into two parts: the knowledge base of the $\text{IDEF}_0$ syntax for an activity box and the knowledge base of $\text{IDEF}_0$ syntax for boundary arrows. The former includes the $\text{IDEF}_0$ syntax rules and goals about an activity box focused on *ACTIVITY NAME, ACTIVITY NUMBER, INPUT, OUTPUT,* and *MECHANISM.* The latter is made up of the $\text{IDEF}_0$ syntax rules and goals about boundary arrows focused on boundary *INPUT, OUTPUT, CONTROL,* and *MECHANISM.* Since the $\text{IDEF}_0$ syntax has semantic meanings, it is difficult to represent completely the $\text{IDEF}_0$ syntax to the knowledge base. For example, *ACTIVITY NAME* should be used in the form of a verb, which needs rules as many as the number of verbs in the dictionary.

In the previous section, *$\text{IDEF}_0$ Diagram Translator*, the reasons why two different predicate data files are generated separately have been discussed. Since the knowledge base should be consistent with the $\text{IDEF}_0$ Diagram Translator, the knowledge base for IDEF$_0$ Syntax Expert System should be separated into two sub-components. It is also necessary to separate it into two sub-components in order to reduce the complexity and redundency of the knowledge base because while checking IDEF$_0$ syntax for an activity box, the knowledge base for boundary arrows is unnecessary. The names of the two separated portions of the knowledge base are the *ActivitySArule* for the activity IDEF$_0$ syntax rules, and the *BoundarySArule* for the boundary arrow IDEF$_0$ syntax rules. A detailed discussion of IDEF$_0$ syntax rules follows in Chapter V.

*Data Base (Working Memory).* The data base contains a broad range of information about the current status of the problem being solved. The temporary output files of the IDEF$_0$ Diagram Translator become the initial data base for the IDEF$_0$ Syntax Expert System in accordance with checking IDEF$_0$ syntax of the activity box or the boundary arrow. While checking the IDEF$_0$ syntax, the data base also contains a list of rules that have been examined and fired. After checking IDEF$_0$, the sequence of the rules fired can be given in order to explain the reasoning process.

*User Interface.* The user interface allows the user to communicate with the expert system and also provides the user with an insight into the problem-solving process carried out by the inference engine. There are several ways to communicate with the expert system such as *questions* and *answers*, *menu choices*, *statements*, and others. In the ISES, the user interface facility as a piece of software is contained in the inference engine and provides a means of asking questions and answering through the *verbose* and *why* trace operation. It also provides menu choices in order to check IDEF$_0$ syntax for activity box or boundary arrow.

### *Testing Techniques*

There are two common techniques to test software referred to as *black box* and *white box* testing (12:470).

Black box testing enables the software engineer to derive sets of input conditions that will fully test all functional requirements for a program. This attempts to find errors in the following categories:

- incorrect or missing functions,
- interface errors,
- errors in data structures,
- performance errors, and
- initialization and termination errors (12:484).

White box testing is a test method based on the control structure of the procedural design. This is used to derive the following test cases:

- guarantee that all independent paths within a module have been exercised at least once,
- exercise all logical decisions on their true and false sides,
- execute all loops at their boundaries and within their operational bounds, and
- exercise internal data structures to assure their validity (12:472).

As mentioned earlier, the black box testing method is useful at this point because it focuses on the functional requirements of the software. The functions defined in the requirements analysis phase are compared to the requirements specification to be sure that all requi·ements are satisfied. In the case of the IDEF$_0$ Diagram Translator, the black box test can be applied. For example, does the IDT translate IDEF$_0$ diagram into a set of predicate data forms which contains all syntactical information of the IDEF$_0$ diagram? In the case of the IDEF$_0$ Syntax Expert System, the black box test can also applied. For example, does the ISES contains all syntactical rules of IDEF$_0$ diagram comparision with the requirements specification.

### *Summary*

This chapter presented a high level software design for the $IDEF_0$ Diagram Translator and the $IDEF_0$ Syntax Expert System. To be consistent with the earlier version of IDT, several considerations of the earlier version are addressed. The screen layout and the menu system modified were presented. In addition to, the main functions of the $IDEF_0$ Diagram and the components of the $IDEF_0$ Syntax Expert are also addressed, and the test design was introduced. The next chapter presents low level design, implementation, and software test for the $IDEF_0$ Diagram Translator and the $IDEF_0$ Syntax Expert System.

# *V. Low Level Design and Implementation*

## *Introduction*

This chapter discusses the low level design and implementation of an IDEF$_0$ Diagram Translator and an IDEF$_0$ Syntax Expert System specified in the previous design chapter. The IDT becomes a portion of the SAtool and translates IDEF$_0$ diagrams into the predicate data forms. These predicate data forms are used for checking IDEF$_0$ syntax and become the initial data base (working memory) of the IDEF$_0$ Syntax Expert System.

## *IDEF$_0$ Diagram Translator Design*

The flow diagram model of the IDEF$_0$ Diagram Translator is shown in Figure 5.1. There are three components in Figure 5.1: Translate Activity, Translate Boundary, and Save Diagram for the IDT. These three components accept an IDEF$_0$ diagram, User's Choice, and Parent Predicate as inputs and generate outputs such as Activity Predicate, Boundary Predicate, and Diagram Predicate applied by Translation Rules. The function of Translate Activity is to translate an activity box in any IDEF$_0$ diagram into the predicate data forms and produce a file of the predicate data forms. The file name of the predicate data forms for an activity box is CHECKBOX.PRO (temporary file). The function of Translate Boundary is to translate boundary information in any IDEF$_0$ diagram and the parent diagram related to the current IDEF$_0$ diagram into the predicate data forms and generate a file of the predicate data forms which is a temporary file. The file name of the predicate data forms for the boundary information is CHECKBOUNDARY.PRO (temporary file). The function of Save Diagram is to translate the IDEF$_0$ diagram into the predicate data forms and save into a file of *.pro. The symbol * is a name which the user enters. The *.pro file is used to check IDEF$_0$ syntax about the boundary arrows. Appendix B shows the structure charts for the IDEF$_0$ Diagram Translator implementation.

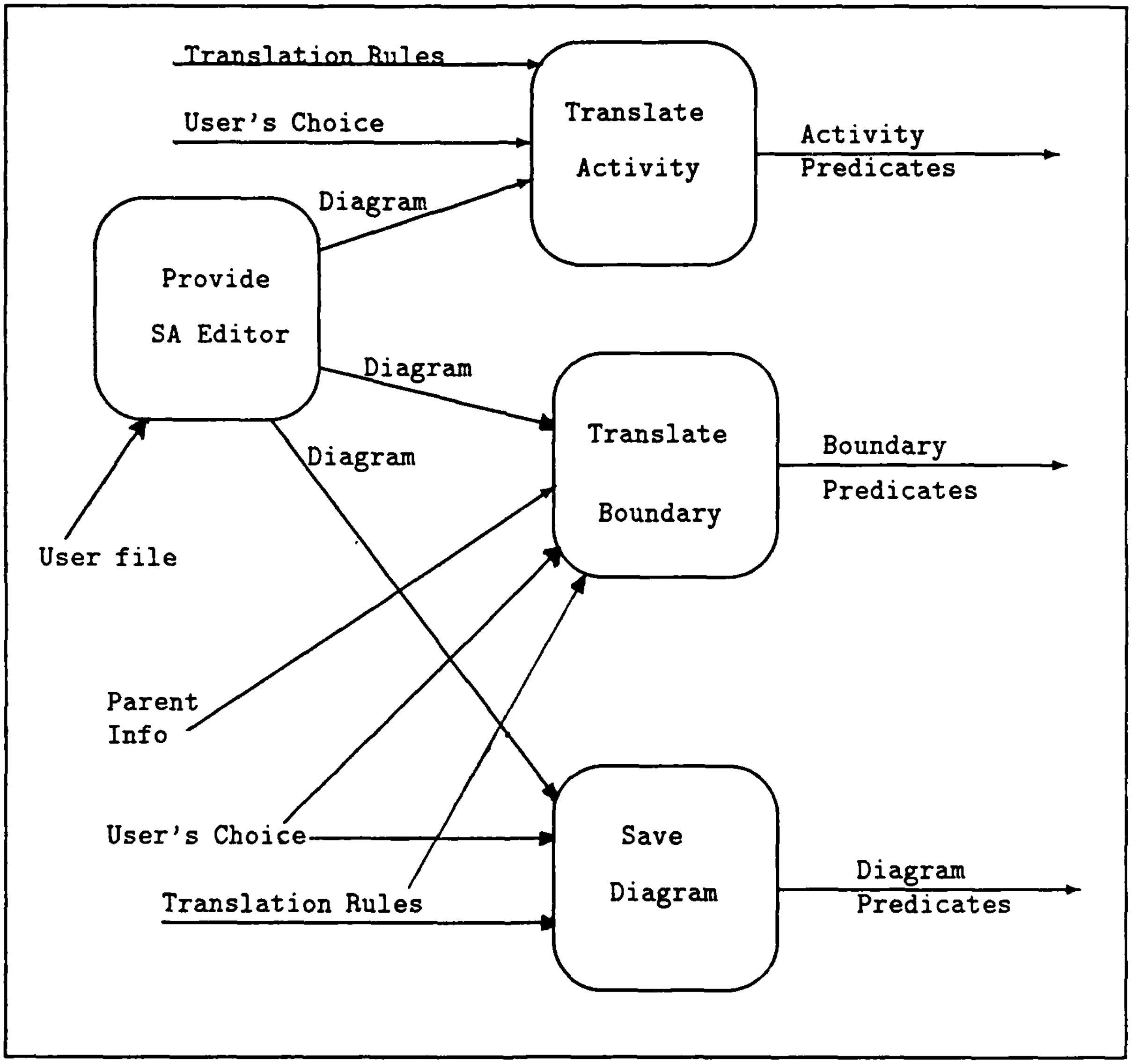


Figure 5.1. Flow Diagram for IDEF$_0$ Diagram Translator

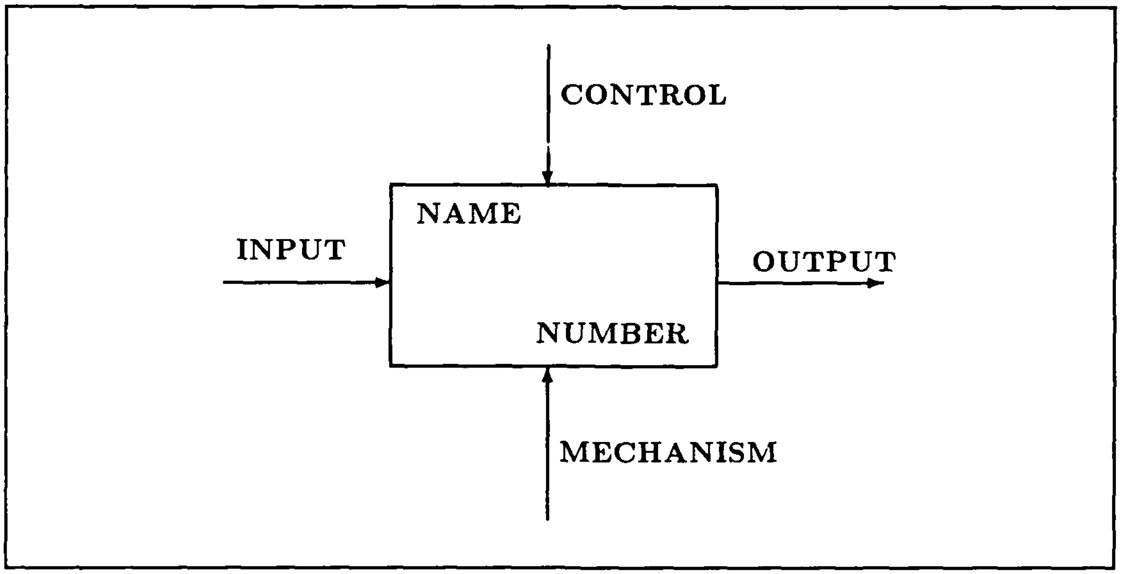


Figure 5.2. A Typical Activity Box

The earlier version of the SAtool was implemented in C. To reuse many modules of the earlier SAtool without modifications or with slight modifications, a decision was made to proceed using C language for the IDEF$_0$ Diagram Translator.

*Translation Rules*

Since the IDEF$_0$ Diagram Translator consists of three components, it is reasonable to discuss the translation rules according to Activity and Boundary.

*Activity.* A typical drawing model for an activity box is shown in Figure 5.2. The IDEF$_0$ Diagram Translator should produce Activity Predicate data forms which include all graphical information of an activity box. As discussed in Chapter IV, the information of an arbitrary activity box, which is based on NAME, NUMBER, INPUT, OUTPUT, CONTROL, and MECHANISM is shown in Figure 5.2. Among the information, the NAME is a key information because the other information depends on the NAME. For example, if the NAME is known among activity boxes, the other information can be extracted easily. Table 5.1 presents the translation

| Term | Predicate | Triple |
|---|---|---|
| NAME | activityname(is, NAME) | [activityname, is, NAME] |
| NUMBER | NAME(number_is, NUMBER) | [NAME, number_is, NUMBER] |
| INPUT | NAME(input_is, LABEL) | [NAME, input_is, LABEL] |
| OUTPUT | NAME(output_is, LABEL) | [NAME, output_is, LABEL] |
| CONTROL | NAME(control_is, LABEL) | [NAME, control_is, LABEL] |
| MECHANISM | NAME(mechanism_is, LABEL) | [NAME, mechanism_is, LABEL] |
| NUMBER OF INPUTS | NAME(has_input_number, COUNT) | [NAME, has_input_number, COUNT] |
| NUMBER OF OUTPUTS | NAME(has_control_number, COUNT) | [NAME, has_output_number, COUNT] |
| NUMBER OF CONTROLS | NAME(has_control_number, COUNT) | [NAME, has_control_number, COUNT] |
| NUMBER OF MECHANISMS | NAME( has_mechanism_number, COUNT) | [NAME, has_mechanism_number, COUNT] |

Table 5.1. Translation Rules for Activity Box

rules for an activity box. Column 1 in the table 5.1 presents the information items of an activity box. Column 2 shows predicate data forms of the items. Since the predicate data forms produced by IDEF$_0$ Diagram Translator become the initial data base of the IDEF$_0$ Syntax Expert System, each item of the predicate data form should be represented by a three-element list of the triple form: [Object, Attribute, Value]. Column 3 shows the triples of the items. The triple form is used as the actual input of the IDEF$_0$ Syntax Expert System. The activity box name, NAME, is translated into the predicate *activityname(is, NAME)*, which means activity box is named as NAME. The predicate *NAME(number_is, NUMBER)* for NUMBER means the number of activity box NAME is NUMBER. In the case of INPUT, it is translated into the predicate *NAME(input_is, LABEL)*, which means the input of activity box NAME is LABEL. Similarly, in the case of OUTPUT, CONTROL, and MECHANISM, they are easily translated into the predicate data forms as shown in the table 5.1. In the case of NUMBER OF INPUTS, it is translated into the predicate *NAME(has_input_number, COUNT)*, which means activity box NAME has COUNT INPUTS. This information is used for checking the number of inputs which is limited and the boundary arrow. Also, NUMBER OF OUTPUTS, NUMBER OF CONTROLS, and NUMBER OF MECHANISMS are translated similarly.

*Boundary.* The type of a boundary arrow is one of either input, output, control, or mechanism. These are the touched arrows of an activity box as well. Since the boundary arrows should be related to the current IDEF$_0$ diagram and an activity box of the parent IDEF$_0$ diagram, the predicate information of the activity box should have been saved already. Table 5.2 shows the translation rules for boundary arrows. BOUNDARY INPUT in column 1 is translated into the predicate *boundary_input(is, LABEL)* in column 2, which means LABEL is a boundary input. The triple [boundary_input, is, LABEL] in column 3 represents the actual data base form for the IDEF$_0$ Syntax Expert System. BOUNDARY OUTPUT, BOUNDARY CONTROL, and BOUNDARY MECHANISM are translated similarly as shown in Table 5.2. NUMBER OF BOUNDARY INPUTS is translated into the predicate *boundary_input(has_number, COUNT)*, which means the number of boundary inputs is COUNT. In the case of NUMBER OF BOUNDARY OUTPUTS, NUMBER OF BOUNDARY CONTROLS, and NUMBER OF BOUNDARY MECHANISMS, they are each translated as shown in Table 5.2.

### *IDEF$_0$ Syntax Expert System*

The detailed structure of the IDEF$_0$ Syntax Expert System is shown in Figure 5.3. Since the user interface portion is contained in the inference engine as discussed in Chapter IV, this is not mentioned in this chapter. Thus, the IDEF$_0$ Syntax Expert System consists of three major components: Rule Base, Working Memory,

| Term | Predicate | Triple |
|---|---|---|
| BOUNDARY INPUT | boundary_input (is, LABEL) | [ boundary_input , is, LABEL ] |
| BOUNDARY OUTPUT | boundary_output (is, LABEL) | [ boundary_output , is, LABEL ] |
| BOUNDARY CONTROL | boundary_control (is, LABEL) | [ boundary_control , is, LABEL ] |
| BOUNDARY MECHANISM | boundary_mechanism (is, LABEL) | [ boundary_mechanism , is, LABEL ] |
| NUMBER OF BOUNDARY INPUTS | boundary_input (has_number, COUNT) | [ boundary_input , has_number, COUNT ] |
| NUMBER OF BOUNDARY OUTPUTS | boundary_output (has_number, COUNT) | [ boundary_output , has_number, COUNT ] |
| NUMBER OF BOUNDARY CONTROLS | boundary_control (has_number, COUNT) | [ boundary_control , has_number, COUNT ] |
| NUMBER OF BOUNDARY MECHANISMS | boundary_mechanism (has_number, COUNT) | [ boundary_mechanism , has_number, COUNT ] |

Table 5.2. Translation Rules for Boundary Arrows

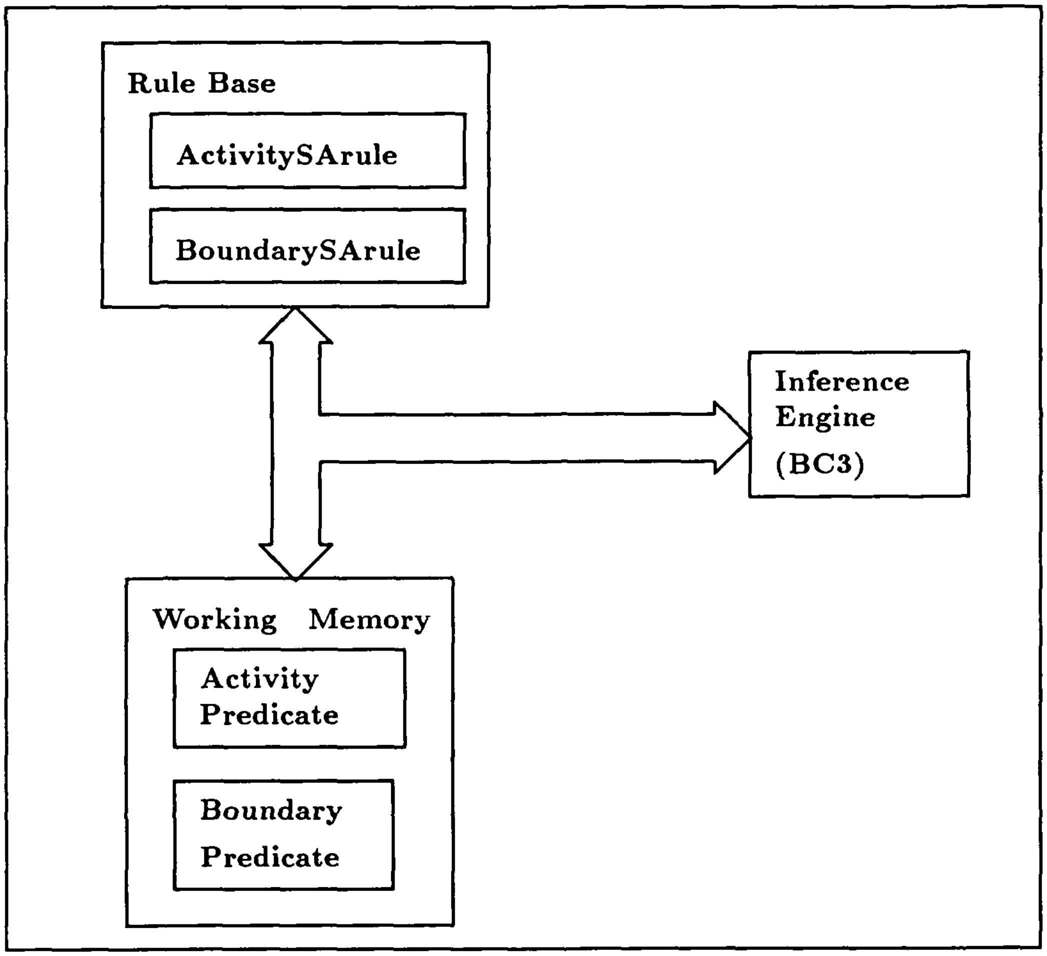


Figure 5.3. Structure of IDEF$_0$ Syntax Expert System

and Inference Engine as shown in Figure 5.3.

The detailed discussion of the Rule Base is given in the next section, *Rule Base.* The Working Memory (Data Base) is initially set up by the predicate output of the IDEF$_0$ Diagram Translator. The predicate output is a collection of fact tripples discussed in the previous section, *Translation Rules.* The inference engine, BC3, which directed problem-solving processes and acted as a rule interpreter, was available. BC3 is a *shell* for a backward chaining expert system. Since the backward chaining strategy is good when there are many more facts than final goals, BC3 is suitable for use as the inference engine of the IDEF$_0$ Syntax Expert System. Also, since BC3 was originally used on the Zenth Z-248 workstations, in order to run it on the Sun workstations, BC3 should be modified. The modified BC3 is listed in appendix F.

BC3 was implemented in Prolog-1 which is a dialect of many Prolog languages and is for the personal computer. To reuse BC3 with slight modification, Quintus Prolog, which is available on the Sun workstations, was selected to implement the inference engine for the ISES.

### *Rule Base*

The inference engine (BC3) applies each element of the rule base to the solution of the specific domain. The specific domain is divided into two categories, one for an activity box and the other for the boundary arrows in any IDEF$_0$ diagram. Thus the rule base (IDEF$_0$ Syntax) is separated into two parts: Activity IDEF$_0$ Syntax and Boundary Arrow IDEF$_0$ Syntax.

*Activity IDEF$_0$ Syntax.* The Activity IDEF$_0$ Syntax focuses on an activity box in any IDEF$_0$ diagram as shown Figure 5.2. The following list in English sentences is extracted from Figure 5.2 for Activity IDEF$_0$ Syntax.

- An activity box must have a name.
- An activity box must have a number except for the top-most level activity box.
- An activity box must have at least a touched control arrow and a touched output arrow.
- If an activity box has touched arrows, the arrows must have their arrow labels.
- If an activity box lies in the top-most level, the box number must be empty.
- If an activity box is not in the top-most level, the box number must be within 1 to 6.

- In the case of CONTROL and OUTPUT, the number of touched arrows must be within 1 to 5.
- In the case of INPUT and MECHANISM, the number of touched arrows must be within 0 to 5.

Figures 5.4 and 5.5 show a more detailed Activity IDEF$_0$ Syntax according to above English sentences.

Column 1 in Figure 5.4 and 5.5 shows the cases of INPUT, OUTPUT, and NUMBER in the figure 5.2. In the case CONTROL and MECHANISM, if-then rules are similar to OUTPUT and INPUT respectively. Column 2 presents all the diagram types which the user could possibly draw. Column 3 shows if-then rules related to the possible drawings. As shown in Figure 5.4, Activity IDEF$_0$ Syntax focuses on whether NAME, INPUT, OUTPUT, CONTROL, MECHANISM, and NUMBER are correct or not. Thus the goals for Activity IDEF$_0$ Syntax rules become a list of triples about NAME, INPUT, OUTPUT, CONTROL, MECHANISM, and NUMBER.

*Boundary IDEF$_0$ Syntax.* The Boundary IDEF$_0$ Syntax is associated with boundary arrows of an IDEF$_0$ diagram and its parent IDEF$_0$ diagram. The following english sentences represent the Boundary IDEF$_0$ Syntax.

- There must be an activity box in the parent IDEF$_0$ diagram.
- The number of input, output, control, or mechanism arrow(s) of the parent activity box must be equal to that of the boundary input, output, control, or mechanism arrow(s).
- Each arrow of the parent activity box must have its label.
- Each boundary arrow must have its label.
- Each boundary arrow label must correspond with label of the parent activity box arrow.

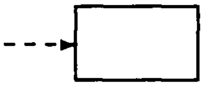
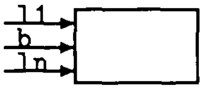
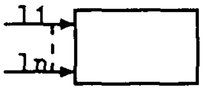
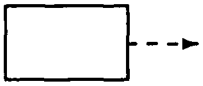
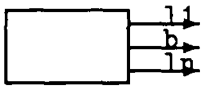
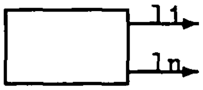
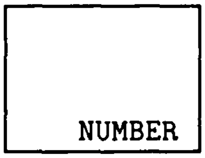
| Case | Diagram | If-then rules |
|---|---|---|
| INPUT (MECHANISM) |  | If there is no input arrow then INPUT is correct on the activity box. |
| |  | If there is at least a blank LABEL then there is a LABEL error on the activity box. |
| |  | If the number of input arrows is greater than 5 then the number of input arrows should be reduced. Otherwise, INPUT is correct. |
| OUTPUT (CONTROL) |  | If there is no output arrow then there is an OUTPUT error on the activity box. |
| |  | If there is at least a blank LABEL then there is a LABEL error on the activity box. |
| |  | If the number of output arrows is greater than 5 that of output arrows should be reduced. Otherwise, OUTPUT is correct. |
| NUMBER |  | If the activity box is the top-most level and NUMBER is empty then NUMBER of the activity box should be empty. |
| | | If the activity box is the top-most level and NUMBER is not empty then NUMBER of the activity box should be empty. |
| | | If the activity box is not the top-most level and NUMBER is greater than 0 and less than 7 then NUMBER of the activity box is correct. Otherwise, NUMBER of the activity box is beyond the limitation. |

Figure 5.4. Activity IDEF$_0$ Syntax

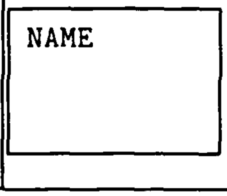
| Case | Diagram | If-then rules |
|---|---|---|
| NAME |  | If there is no activity box name then there is a NAME error on the activity box. |
| | | If there is a NAME on the activity box then NAME is correct. |

Figure 5.5. Activity IDEF$_0$ Syntax(continued)

- *In the case of* (boundary) CONTROL and (boundary) OUTPUT, the number of arrows must be greater than or equal to 1 and less than 6.
- In the case of (boundary) INPUT and (boundary) MECHANISM, the number of arrows must be within 0 to 5.

Figure 5.6 represents the more detailed Boundary IDEF$_0$ Syntax. Column 1 in Figure 5.6 and 5.7 shows the cases of Boundary Input and Boundary Output. If-then rules for Boundary Mechanism and Boundary Control are similar to Boundary Input and Boundary Output respectively. Column 2 shows the models of a parent activity box which is possibly drawn focused on INPUT(OUTPUT) arrow(s). Column 3 shows the models of Boundary Input(Output) arrow(s) which is possibly drawn. Column 4 presents if-then rules about the Boundary IDEF$_0$ Syntax. Boundary IDEF$_0$ Syntax focuses on whether the boundary arrows in any IDEF$_0$ diagram correspond with the arrows on the parent activity box. Thus the goals for Boundary IDEF$_0$ Syntax rules is a list of triples about Boundary Input, Boundary Output, Boundary Control, and Boundary Mechanism.

### *Software Test*

The software testing methods for IDEF$_0$ Diagram Translator and IDEF$_0$ Syntax Expert System were performed using three steps: unit testing, integration test-

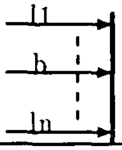
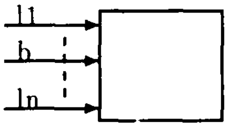
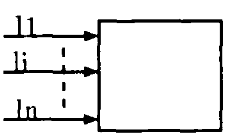
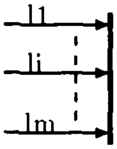
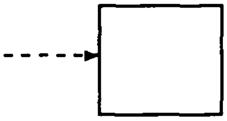
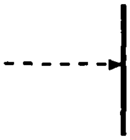
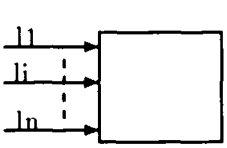
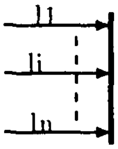
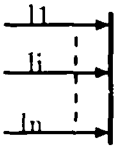
| CASE | PARENT | CHILD | If-then rules |
|---|---|---|---|
| PARENT | THERE IS NO PARENT BOX INFORMATION. | DON'T CARE | If there is no information about the parent activity box then can not check. |
| BOUNDARY INPUT (MECHANISM) | DON'T CARE |  | If there is at least one blank boundary arrow LABEL then there is a LABEL error. |
| |  | DON'T CARE | If there is at least one blank LABEL on the parent activity box then parent Input arrow has no LABEL. |
| |  |  | If the number of Input arrows of the parent box is greater than or less than that of the boundary Input arrows then there is an error of the number of parent Input and boundary Input arrows. |
| |  |  | If there is no Input arrow of parent activity box and boundary arrow then boundary Input is correct. |
| |  |  | If LABELs of Input arrows of the parent activity box and boundary Input arrows are all matched then boundary Input is correct. |
| | | | If there is at least one LABEL which is mismatched then there is an error of mismatched LABEL. |
| | DON'T CARE |  | If the number of boundary Input arrows is greater than 5 then the number of those should be reduced. |

Figure 5.6. Boundary IDEF$_0$ Syntax

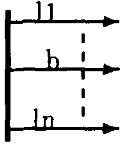
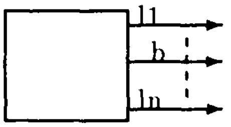
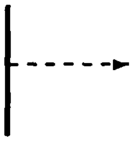
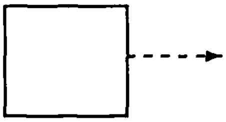
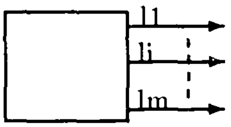
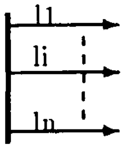
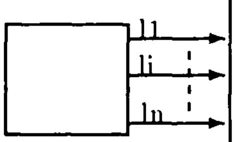
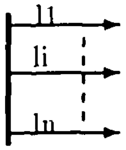
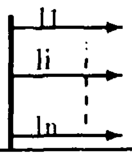
| CASE | PARENT | CHILD | If-then rules |
|---|---|---|---|
| BOUNDARY OUTPUT (CONTROL) | DON'T CARE |  | If there is at least one blank boundary arrow LABEL then there is a LABEL error. |
| |  | DON'T CARE | If there is at least one blank LABEL on the parent activity box then parent Output arrow has no LABEL. |
| | DON'T CARE |  | If there is no boundary Output arrow then there must be at least a boundary Output arrow. |
| |  | DON'T CARE | If there is no Output arrow then there must be at least an Output arrow on parent activity box. |
| |  |  | If the number of Output arrows of the parent box is greater than or less than that of the boundary Output arrows then there is an error of the number of parent Output and boundary Output arrows. |
| |  |  | If LABELs of Input arrows of the parent activity box and boundary Input arrows are all matched then boundary Input is correct. |
| | | | If there is at least one LABEL which is mismatched then there is an error of mismatched LABEL. |
| | DON'T CARE |  | If the number of boundary Output arrows is greater than 5 then the number of those should be reduced. |

Figure 5.7. Boundary $IDEF_0$ Syntax(Continued)

ing, and validation testing (12:502). These methods use the white box testing methods discussed in Chapter IV.

The Unit testing step focuses on each module individually to be sure that it functions properly as a unit (12:502). The test considerations are module interface, data structure, boundary conditions, basis path through the control structure, and error handling paths (12:503). Unit test considerations are applied to IDT and ISES individually. For IDEF$_0$ Diagram Translator, IDEF$_0$ diagram is prepared. Predicate data files are also prepared for IDEF$_0$ Syntax Expert System.

The Integration testing step is applied to take unit-tested modules and construct a complete program structure to ensure that the interfaces between modules are correct (12:507). Bottom-up integration is used because IDEF$_0$ Diagram Translator and IDEF$_0$ Syntax Expert System are lower-level than SAtool. First, testing for IDEF$_0$ Diagram Translator was performed and then the whole system of SAtool was examined. Also testing of IDEF$_0$ Syntax Expert System was applied separated because ISES is separated with IDT and SAtool.

The Validation testing step is performed to provide final assurance that the software meets the mentioned requirements. This focuses on the *Are we building the right product?* (12:499). This step was used to examine whether the IDT produced predicate data forms correctly and the ISES checked completely IDEF$_0$ syntax.

### *Summary*

This chapter described the low level design and implementation of IDEF$_0$ Diagram Translator and IDEF$_0$ Syntax Expert System based on the requirements of the tool. The translation rules and the components of ISES are also represented. Finally, testing methodology applied in this investigation are discussed.

# *VI. Conclusions and Recommendations*

### *Introduction*

The objective of this thesis investigation was to design and implement an application of expert system for checking IDEF$_0$ syntax of IDEF$_0$ diagrams as derived from the SAtool. This chapter presents the conclusions and directions for possible future research.

### *Conclusions*

This investigation is classified into two major categories: IDEF$_0$ Diagram Translator and IDEF$_0$ Syntax Expert System. The work for the IDEF$_0$ Diagram Translator was performed in two phases. During the first phase, the formulation of graphical features of the IDEF$_0$ diagram was derived through predicate calculus representation, since the predicate calculus is a *convenient representation* for facts and rules of inference. The formal definition of the IDEF$_0$ graphical features does not have completeness but consistency because the IDEF$_0$ graphical language contains semantic meanings. The second phase included the development of the IDEF$_0$ Diagram Translator which translates the IDEF$_0$ graphical features in the IDEF$_0$ diagram into a set of predicate data forms. The predicate data forms focus on an activity box and associated arrows and on the boundary arrows in any IDEF$_0$ diagram. Predicate data forms become the data base (working memory) for the IDEF$_0$ Syntax Expert System. The IDEF$_0$ Syntax Expert System consists of the inference engine, the knowledge base, the data base, and the user interface. The inference engine applies the knowledge to the solution of a specific domain. To check IDEF$_0$ syntax in any IDEF$_0$ diagram, the backward chaining control strategy is useful because there are many more facts than final goals. The knowledge base was identified with emphasis on the activity box and on the boundary arrows in any IDEF$_0$ diagram. The knowledge base structure is easy to extend to new IDEF$_0$ syntax rules independently of other rules and to change independently of other rules. Each segment of the knowledge base defines a small and relatively independent piece of knowledge.

*Recommendations*

Based on the results of this study and the observations made during it, this section presents some recommendations for future research which could lead to enhance the capability of the ISES.

*Activity* Currently, the tool can check the $IDEF_0$ syntax of only a single activity box and associated arrows in any $IDEF_0$ diagrams. The relationship among activity boxes and arrows in composite $IDEF_0$ diagrams could be defined to enhancements of the ISES's capability. For example,

- The name of an activity box should not be the same as that of other activity boxes in any $IDEF_0$ diagrams.
- The number of an activity box should not be the same as that of other activity boxes in any $IDEF_0$ diagram.
- The line label on an activity box should not be the same as that on other activity boxes in any $IDEF_0$ diagram.

*Boundary* Since the ISES can check syntax of the boundary arrows except the tunnel arrow, add the $IDEF_0$ syntax rules about the tunnel arrow. This issue implies $IDEF_0$ syntax check of the multilevel $IDEF_0$ diagrams.

Integrate the translation process with the syntax checking process to be more user friendly. This issue needs to address how the C language should interface with Quintus Prolog or some other prolog.

Apply the structure of the knowledge-based $IDEF_0$ syntax system to incorporate the design knowledge of a specific software application. Since the design knowledge provides a means of abstracting software design into resuable modules, the design knowledge using the IDEF$_0$ methodology can be reused for a similar software design. The knowledge-based IDEF$_0$ syntax system uses IDEF$_0$ model segments to represent design modules which are combined and refined to generate an entire IDEF$_0$ model.

### *Summary*

This chapter presented the conclusions derived from the design and implementation of an application of expert system for checking IDEF$_0$ syntax of IDEF$_0$ diagrams drawn from the SAtool and the recommendations for future research.

# Appendix A. *Requirements Analysis Diagrams*

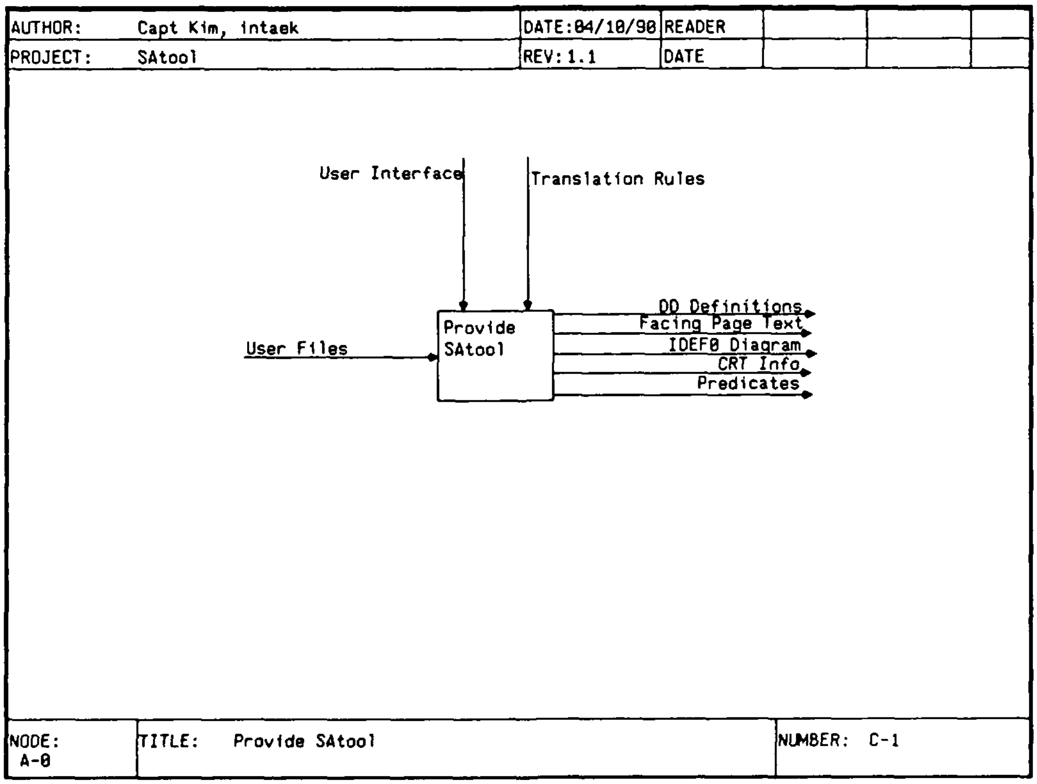
This appendix contains the requirements analysis IDEF$_0$ diagrams for the IDEF$_0$ Diagram Translator. These diagrams are not exactly one-to-one correspondence with the implementation modules, but are close.

## *List of Figures*

Figure Page

A-0 Provide SAtool

Abstract: Provide SAtool provides a means of mechanism by which the user is able to draw Activity IDEF0 Diagrams. From these diagrams, Facing Page Text and Data Dictionaries for Activities and Data and Predicate Data forms are generated.



| | | | | | | |
|---|---|---|---|---|---|---|
| AUTHOR: Capt Kim, intaek | DATE:04/10/90 | READER | | | |
| PROJECT: SAtool | REV: 1.1 | DATE | | | |

| | | |
|---|---|---|
| NODE: A-0 | TITLE: Provide SAtool | NUMBER: C-1 |

Figure A.1. Provide SAtool (Node A-0)

A0 Provide SAtool

Abstract: Provide SAtool provides the user a mechanism by which the user is able to draw Activity IDEF0 Diagrams. From these diagrams, Facing Page Text, Data Dictionaries for Activities and Data, and Predicate Data forms are generated.

A1 Provide SA Editor provides a means of drawing Activity IDEF0 Diagrams. From these diagrams, Facing Page Text and Data Dictionaries for Activities and Data are generated.

A2 Translate Diagram provides a means of translating IDEF0 Diagrams into Predicate Data Forms.

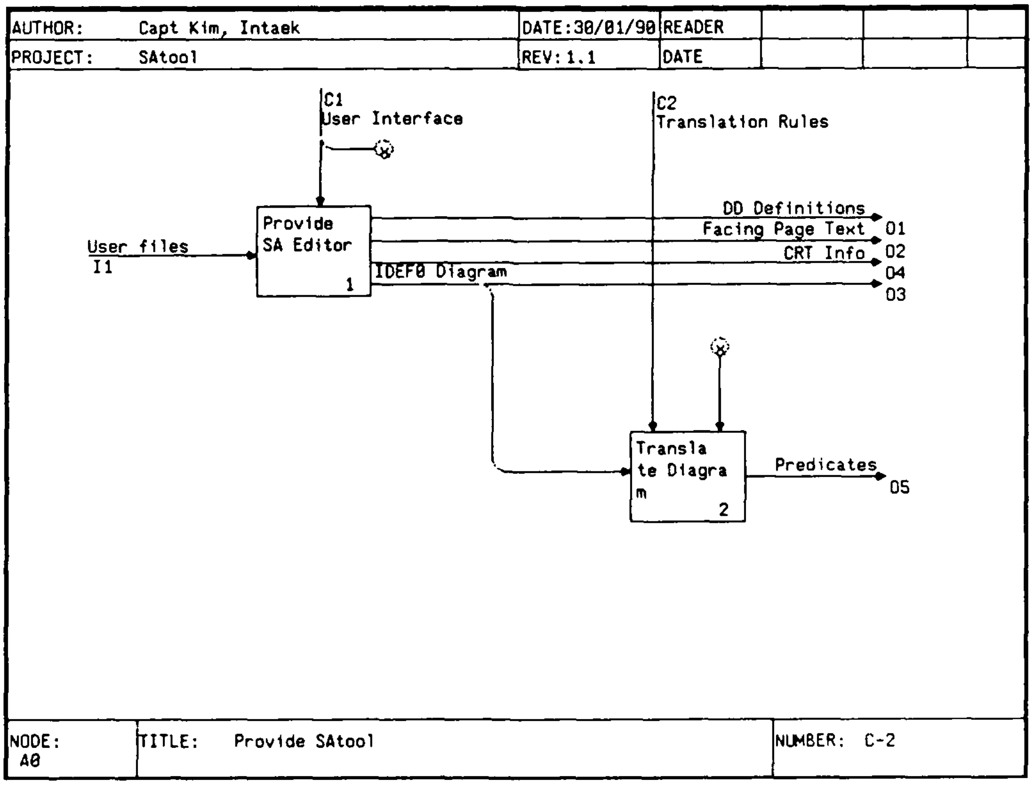


Figure A.2. Provide SAtool (Node A0)

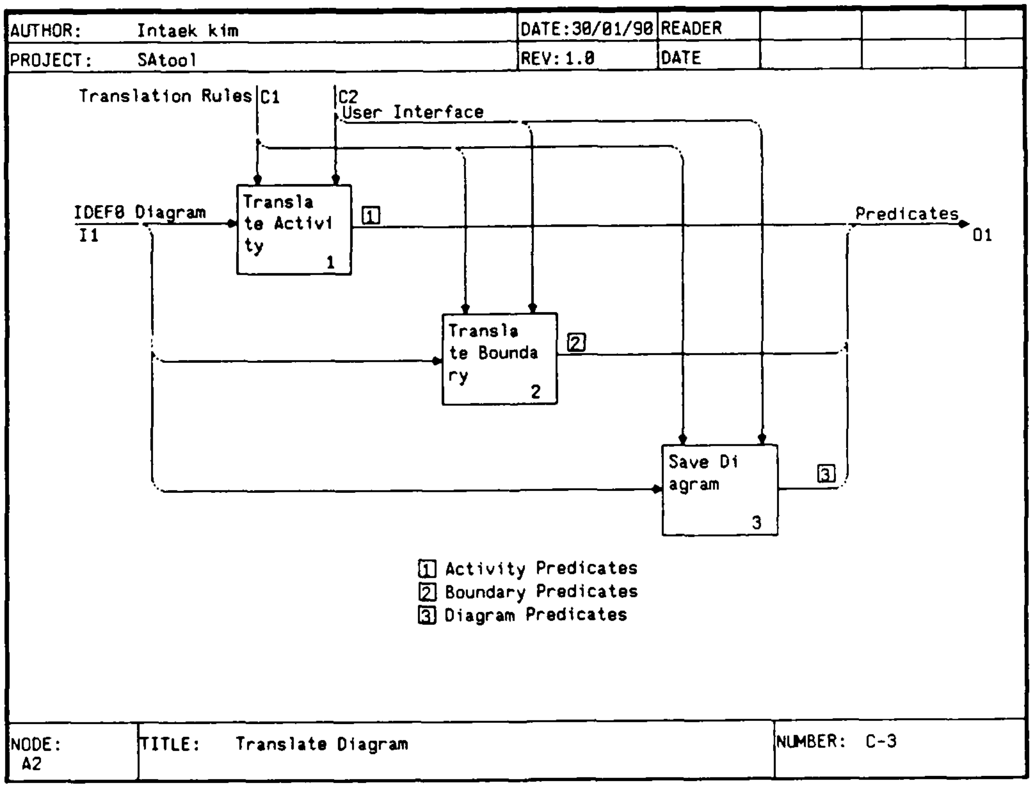
A2 Translate Diagram

Abstract: Translate Diagram provides the user a means of translating IDEF0 Diagrams into Predicate Data Forms.

A21 Translate Activity provides a means of translating an activity box in any IDEF0 Diagrams into a set of predicate data forms through the translation rules of the activity box.

A22 Translate Boundary provides a means of translating the boundary arrows in any IDEF0 Diagrams into Predicate data forms through the translation rules of the boundary arrows.

A23 Save Diagram provides a means of translating the IDEF0 Diagram into a set of the predicate data forms and saving it into a file which user specifies interactively.



|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
| AUTHOR: Intaek kim | DATE:30/01/90 | READER |  |  |  |  |
| PROJECT: SAtool | REV: 1.0 | DATE |  |  |  |  |

NODE: A2

TITLE: Translate Diagram

NUMBER: C-3

Figure A.3. Translate Diagram (Node A2)

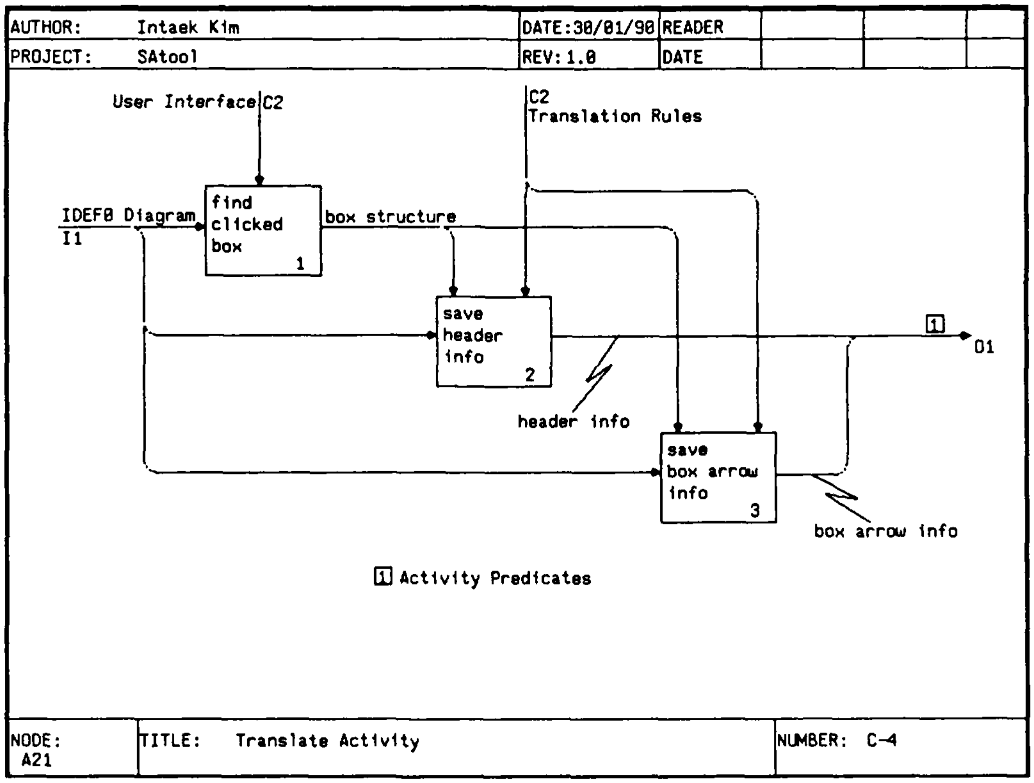
A21 Translate Activity

Abstract: Translate Activity provides a means of translating an activity box in any IDEF0 Diagrams into a set of predicate data forms through the translation rules of the activity box.

A211 'find clicked box' module provides a means of finding an activity box which user specifies using the mouse in the IDEF0 diagram.

A212 'save header info' module provides a means of saving the head information of the IDEF0 diagram which is needed to check Boundary IDEF0 Syntax.

A213 'save box arrow info' module provides a means of grasping the information of the activity box and the arrows which are attatched on the activity box.



|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
| AUTHOR: | Intaek Kim | DATE:30/01/90 | READER |  |  |  |
| PROJECT: | SAtool | REV: 1.0 | DATE |  |  |  |

|  |  |  |
|---|---|---|
| NODE:<br>A21 | TITLE: Translate Activity | NUMBER: C-4 |

Figure A.4. Translate Activity (Node A21)

A213 save box arrow info

Abstract: 'save box arrow info' module provides a means of grasping the information of the activity box and the arrows which are attatched on the activity box.

A2131 'get box info' module provides a means of holding and saving the activity box name and the number into a file in forms of predicate using the translation rules.

A2132 'get arrow info' module provides the information of the arrows which are touched an activity box in forms of predicate.

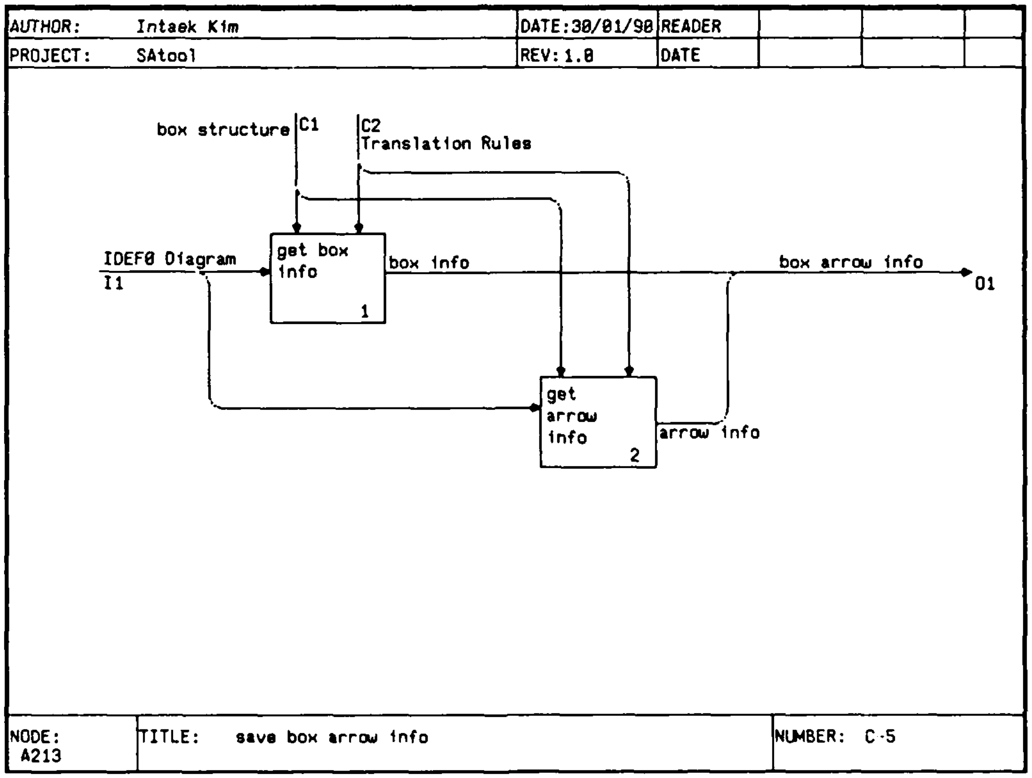


Figure A.5. Save Box Arrow Info (Node A213)

A2132 get arrow info

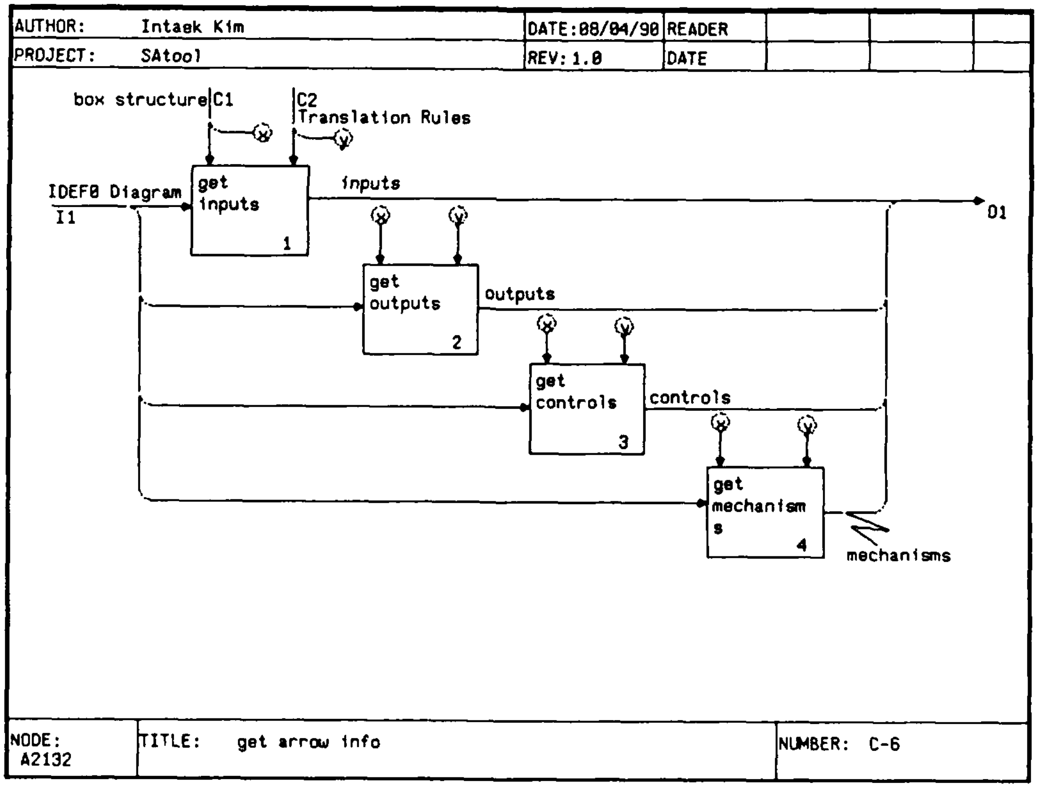
Abstract: 'get arrow info' module provides the information of the arrows which are touched an activity box in forms of predicate.

A21321 'get inputs' module provides a means of grasping the predicate data forms of all kind of input arrows which are touched on an activity box.

A21322 'get outputs' module provides a means of grasping the predicate data forms of all kind of output arrows which are touched on an activity box.

A21323 'get controls' module provides a means of grasping the predicate data forms of all kind of control arrows which are touched on an activity box.

A21324 'get mechanisms' module provides a means of grasping the predicate data forms of all kind of mechanism arrows which are touched on an activity box.



AUTHOR: Intaek Kim
DATE: 08/04/90
READER
PROJECT: SAtool
REV: 1.0
DATE

NODE: A2132
TITLE: get arrow info
NUMBER: C-6

Figure A.6. Get Arrow Info (Node A2132)

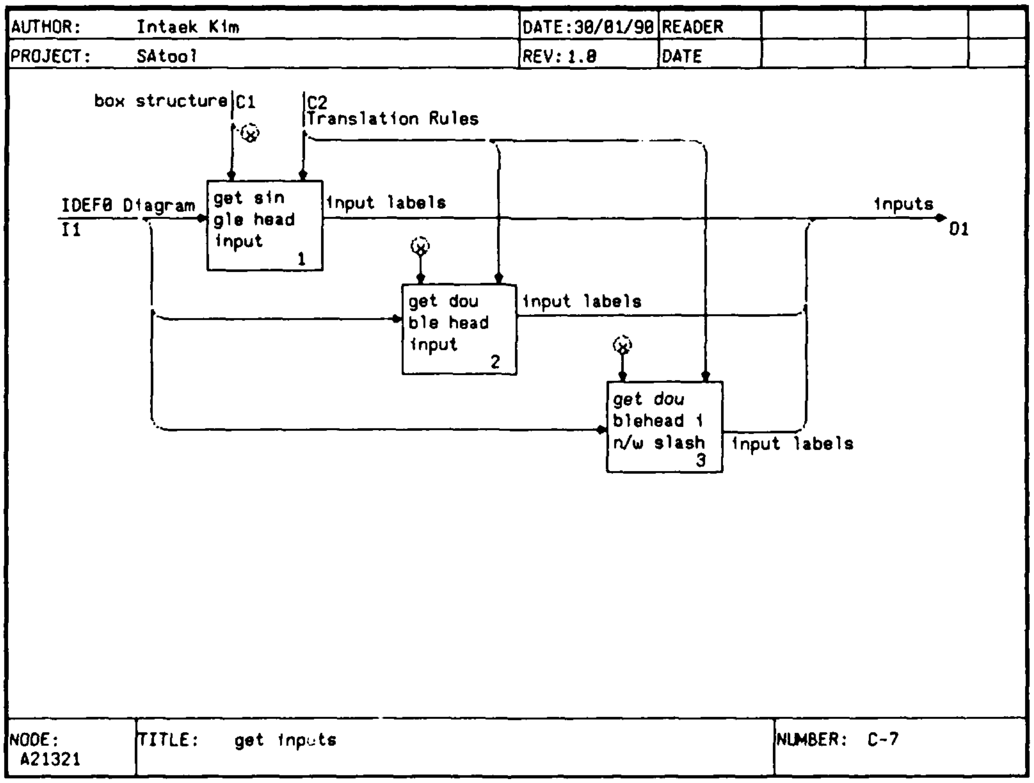
A21321 get inputs

Abstract: 'get inputs' module provides a means of grasping the predicate data forms of all kind of input arrows which are touched on an activity box.

A213211 'get single head input' module provides a means grasping the single headed input arrows' information translated in the predicate data forms which are touched on an activity box.

A213212 'get double head input' module provides a means of grasping the predicate data forms of the double headed input arrows which are touched on an activity box.

A213213 'get doublehead in/w slash' module provides a means of grasping the predicate data forms of the double headed input arrows with the slash which are touched on an activity box.



| | | | | | | |
|---|---|---|---|---|---|---|
| AUTHOR: | Intaek Kim | DATE:30/01/90 | READER | | | |
| PROJECT: | SAtool | REV: 1.0 | DATE | | | |

NODE: A21321 | TITLE: get inputs | NUMBER: C-7

Figure A.7. Get Inputs (Node A21321)

A21322 get outputs

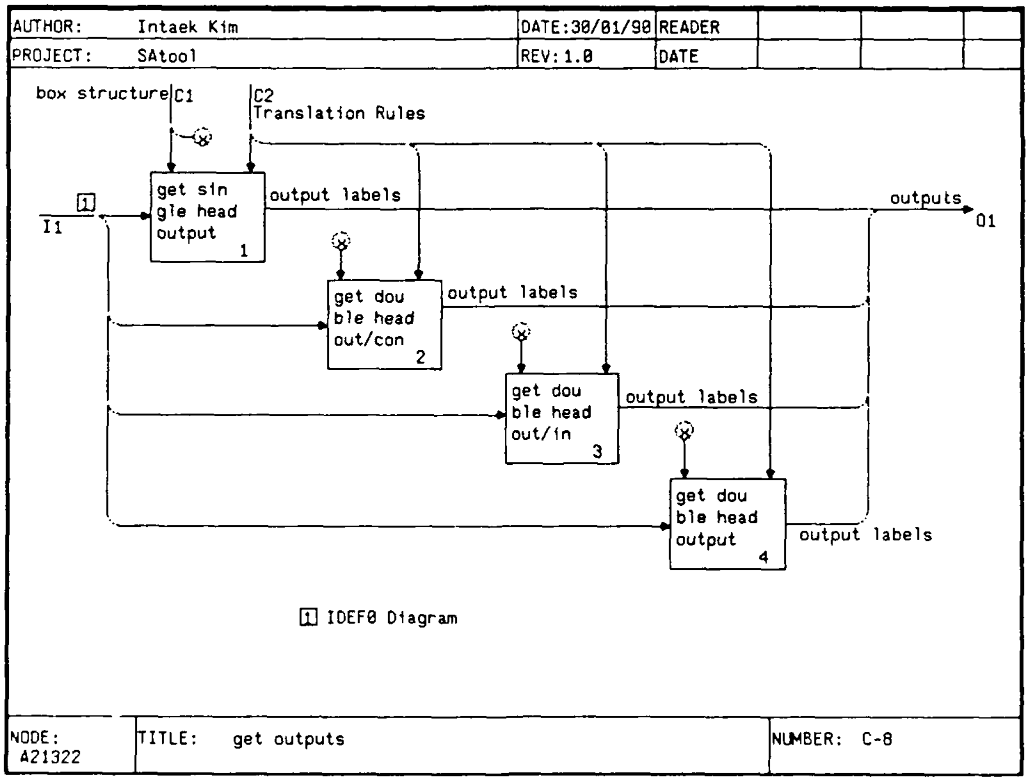
Abstract: 'get outputs' module provides a means of grasping the predicate data forms of all kind of output arrows which are touched on an activity box.

A213221 'get single head output' module provides a means of grasping the predicate data forms of the output arrows with a single head which are touched on an activity box.

A213222 'get double head out/con' module provides a means of grasping the predicate data forms of the output arrows with the double head one for the output and the other for the control arrow of another box.

A213223 'get double head out/in' module provides a means of grasping the predicate data forms of the output arrows with the double head one for the output and the other for the input arrow of another box.

A213224 'get double head output' module provides a means of grasping the predicate data forms of the output arrow which leaves the right side of an activity box and there is a double head.



AUTHOR: Intaek Kim DATE:30/01/90 READER
PROJECT: SAtool REV:1.0 DATE

NODE: A21322 TITLE: get outputs NUMBER: C-8

Figure A.8. Get Outputs (Node A21322)

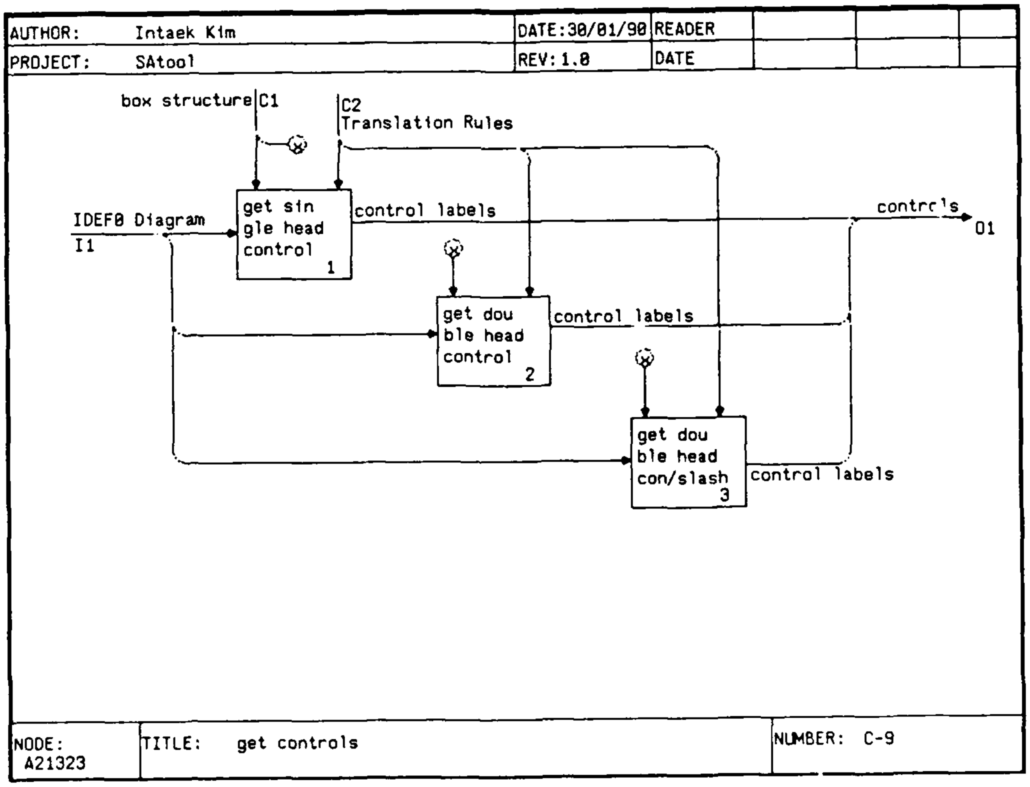
## A21323 get controls

Abstract: 'get controls' module provides a means of grasping the predicate data forms of all kind of control arrows which are touched on an activity box.

A213231 'get single head control' module provides a means of grasping the predicate data forms of the control line which comes to the upper side of an activity box and there is a single headed arrow.

A213232 'get double head control' module provides a means of grasping the predicate data forms of the control line which comes to the upper side of an activity box and there is a double headed arrow.

A213233 'get double head con/slash' module provides a means of grasping the predicate data forms of the control line which comes to the upper side of an activity box and there is a double headed arrow with a slash.



| | | | | | | |
|---|---|---|---|---|---|---|
| AUTHOR: | Intaek Kim | DATE:30/01/90 | READER | | | |
| PROJECT: | SAtool | REV: 1.0 | DATE | | | |

| | | |
|---|---|---|
| NODE: A21323 | TITLE: get controls | NUMBER: C-9 |

Figure A.9. Get Controls (Node A21323)

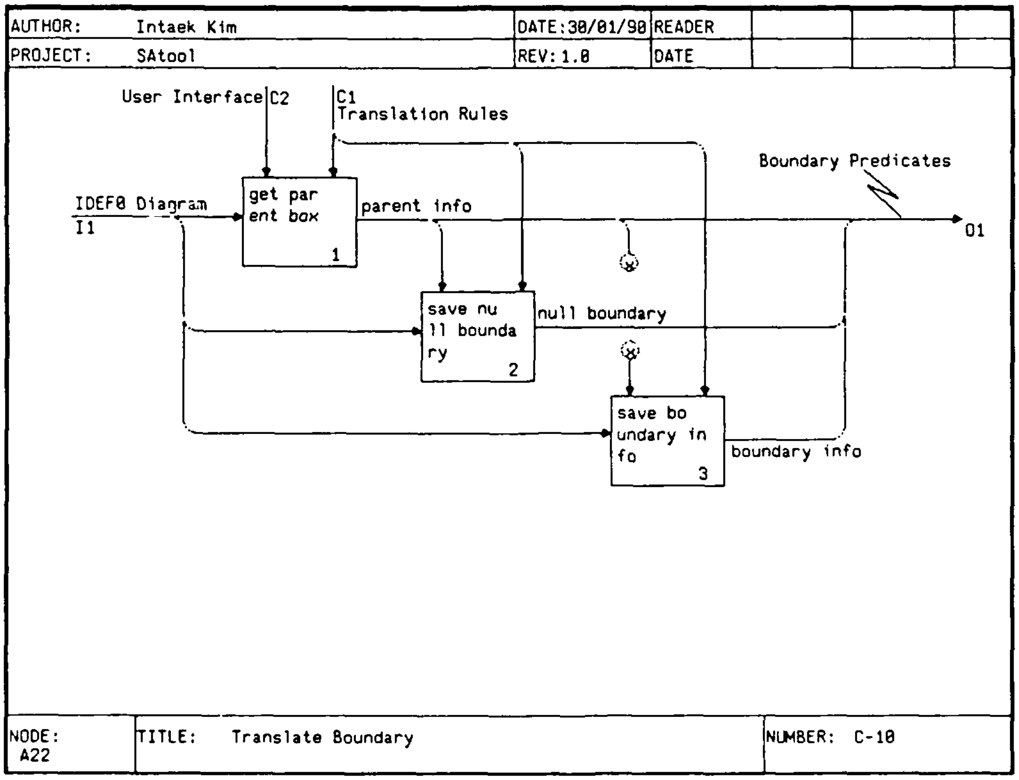
## A22 Translate Boundary

Abstract: Translate Boundary provides a means of translating the boundary arrows in any IDEF0 Diagrams into Predicate data forms through the translation rules of the boundary arrows.

A221 'get parent box' module provides a means of grasping the predicates for the parent activity box and producing the parent information.

A222 'save null boundary' module provides a means of grasping the predicates if there is no the boundary arrow in according with the input, output, control, or mechanism of the IDEF0 diagram.

A223 'save boundary info' module provides a means of grasping the predicates of the boundary arrows if there is at least one boundary arrow in the IDEF0 diagram.



| | | | | | | |
|---|---|---|---|---|---|---|
| AUTHOR: | Intaek Kim | DATE:30/01/90 | READER | | | |
| PROJECT: | SAtool | REV:1.0 | DATE | | | |

| | | |
|---|---|---|
| NODE: A22 | TITLE: Translate Boundary | NUMBER: C-10 |

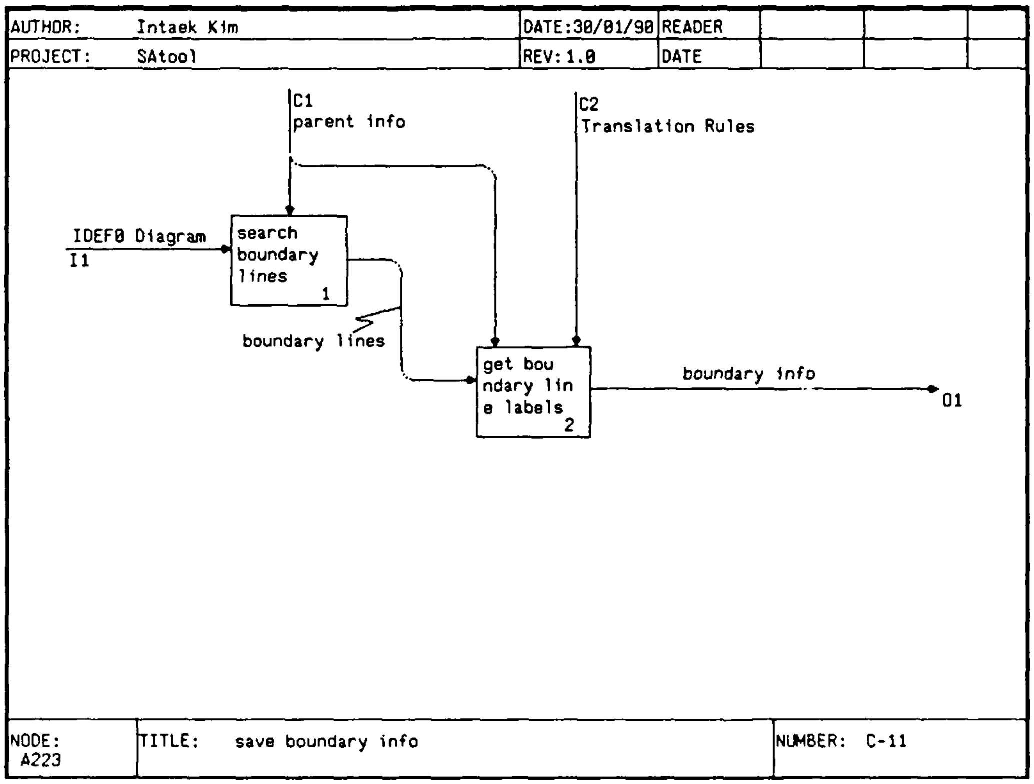
Figure A.10. Translate Boundary (Node A22)

A223 save boundary info

Abstract: 'save boundary info' module provides a means of grasping the predicates of the boundary arrows if there is at least one boundary arrow in the IDEF0 diagram.

A2231 'search boundary lines' module provides a means of searching for the boundary lines for the IDEF0 diagram and producing a linked list of line structure as the output.

A2232 'get boundary line labels' module provides a means of grasping the line labels of the boundary lines in the IDEF0 diagram.



AUTHOR: Intaek Kim DATE:30/01/90 READER

PROJECT: SAtool REV: 1.0 DATE

NODE: A223 TITLE: save boundary info NUMBER: C-11

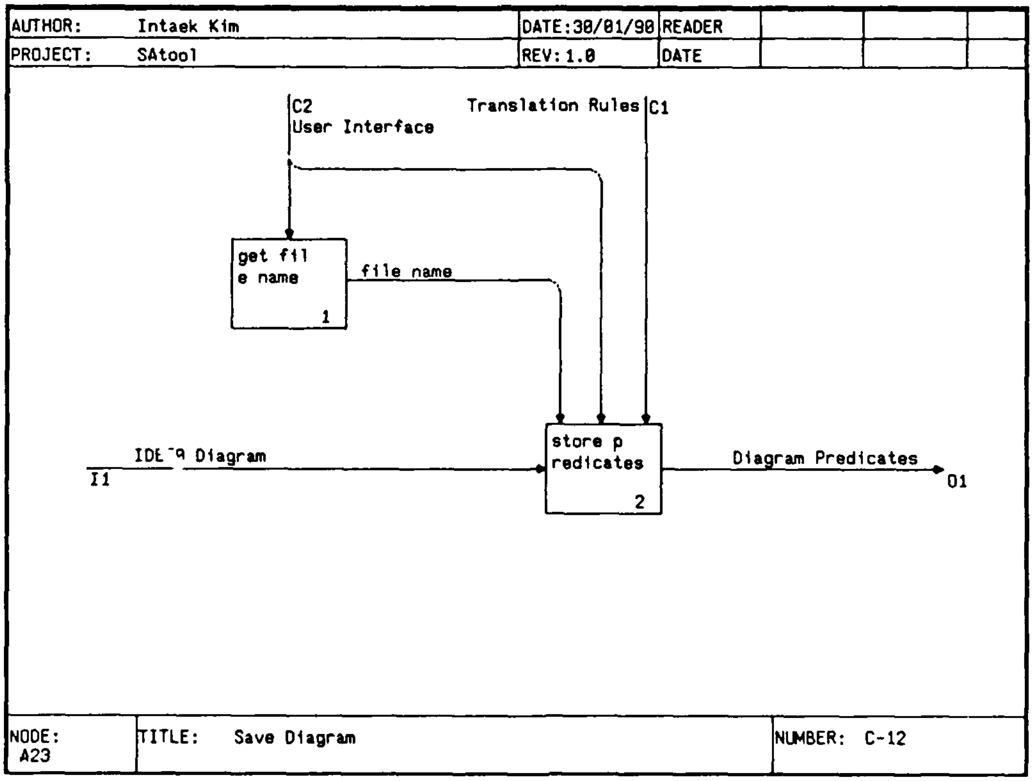
Figure A.11. Save Boundary Info (Node A223)

## A23 Save Diagram

Abstract: Save Diagram provides a means of translating the IDEF0 Diagram into a set of the predicate data forms and saving it into a file which user specifies interactively.

A231 The function of 'get file name' module is to get a file name from the user in order to save the predicate data forms for the IDEF0 diagram into it.

A232 The function of 'store predicates' module is to save the predicates for the IDEF0 diagram into a file which the user specifies.



| | | | | | | |
|---|---|---|---|---|---|---|
| AUTHOR: | Intaek Kim | DATE:30/01/90 | READER | | | |
| PROJECT: | SAtool | REV:1.0 | DATE | | | |

| | | |
|---|---|---|
| NODE:<br>A23 | TITLE: Save Diagram | NUMBER: C-12 |

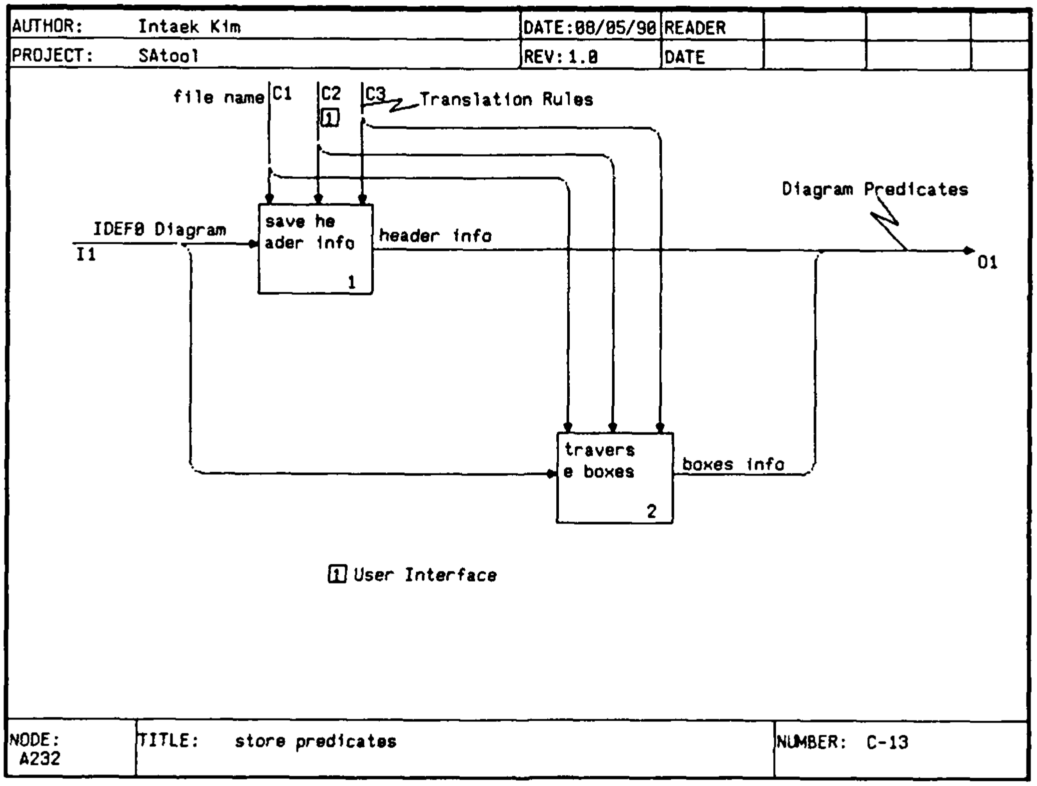
Figure A.12. Save Diagram (Node A23)

A232 store predicates

Abstract: The function of 'store predicates' module is to save the predicates for the IDEF0 diagram into a file which the user specifies.

A2321 'save header info' has the same function of module A212. See A212 description.

A2322 'traverse boxes' module function is to traversing every boxes in the IDEF0 diagram.



| | | | | | | |
|---|---|---|---|---|---|---|
| AUTHOR: | Intaek Kim | DATE:08/05/90 | READER | | | |
| PROJECT: | SAtool | REV:1.0 | DATE | | | |

| | | |
|---|---|---|
| NODE: A232 | TITLE: store predicates | NUMBER: C-13 |

Figure A.13. Store Predicates (Node A232)

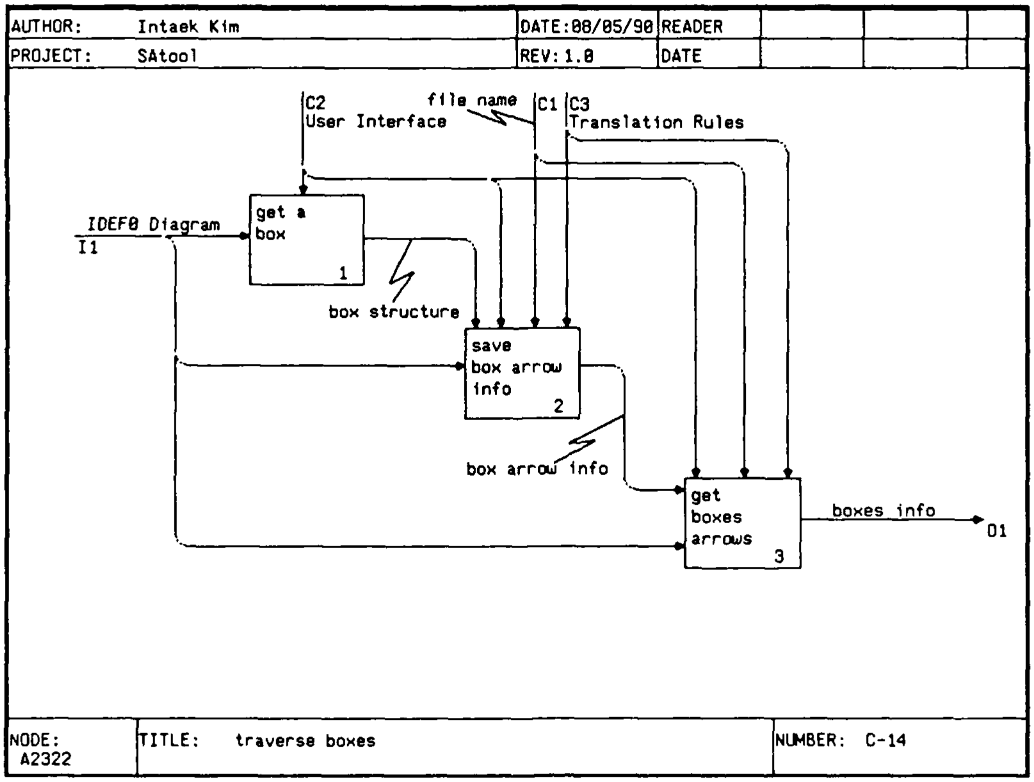
A2322 traverse boxes

Abstract: 'traverse boxes' module function is to traversing every boxes in the IDEF0 diagram.

A23221 'get a box' module function is to get the information for an activity box in the IDEF0 diagram.

A23222 'save box arrow info' module function is the same as module A213.

A23223 'get boxes arrows' module function is to gether the predicates of every activity box and arrow in the IDEF0 diagram.



| | | | | | | |
|---|---|---|---|---|---|---|
| AUTHOR: | Intaek Kim | DATE:08/05/90 | READER | | | |
| PROJECT: | SAtool | REV: 1.0 | DATE | | | |

| | | |
|---|---|---|
| NODE: A2322 | TITLE: traverse boxes | NUMBER: C-14 |

Figure A.14. Traverse Boxes (Node A2322)

# Appendix B. *Structured Chart*

This appendix contains the detailed design structure charts for the IDEF$_0$ Diagram Translator implementation. The detailed design is concerned with the requirements analysis diagrams in appendix A. There is a close, but not exactly one-to-one, correspondence between the design modules and the implementation modules.

## *List of Figures*

Figure Page

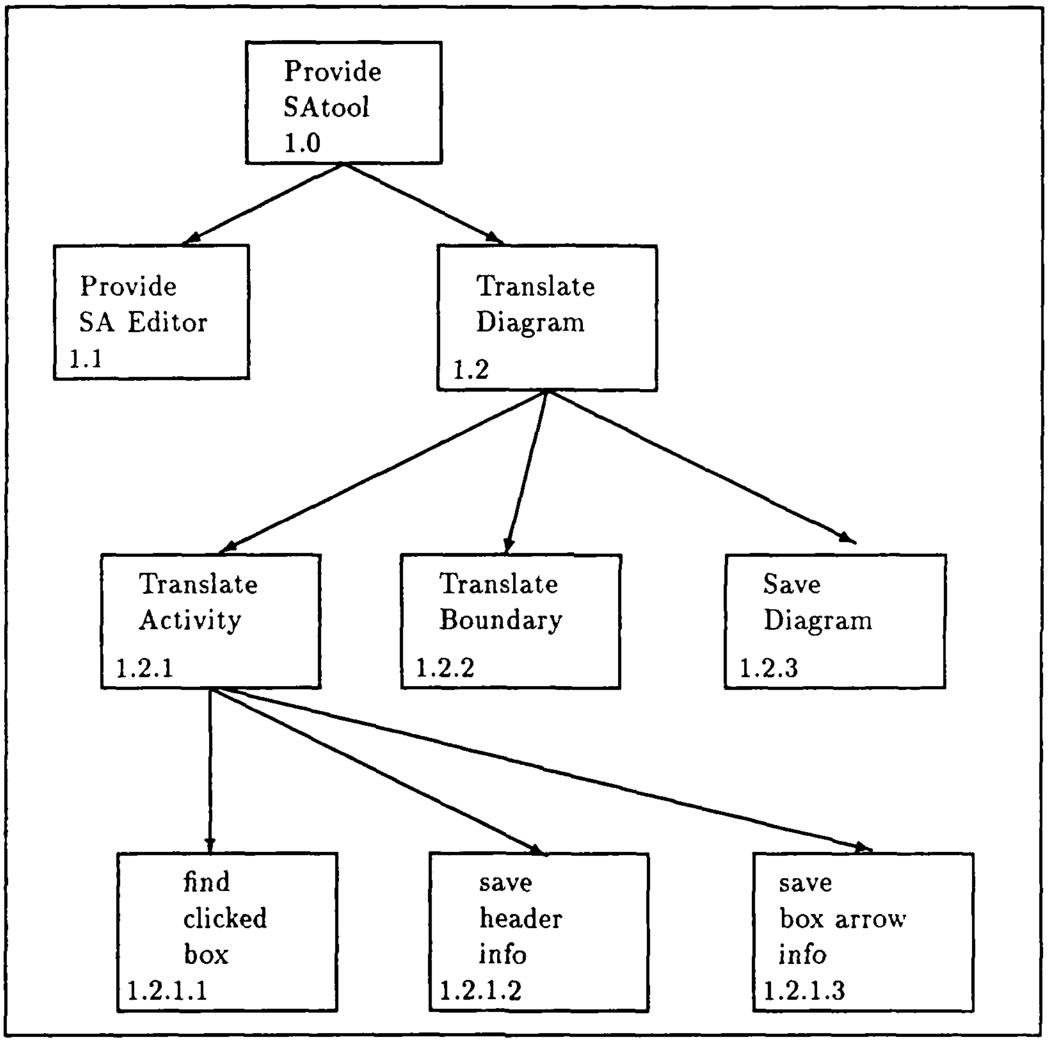


Figure B.1. Provide SAtool(Module 1.0)

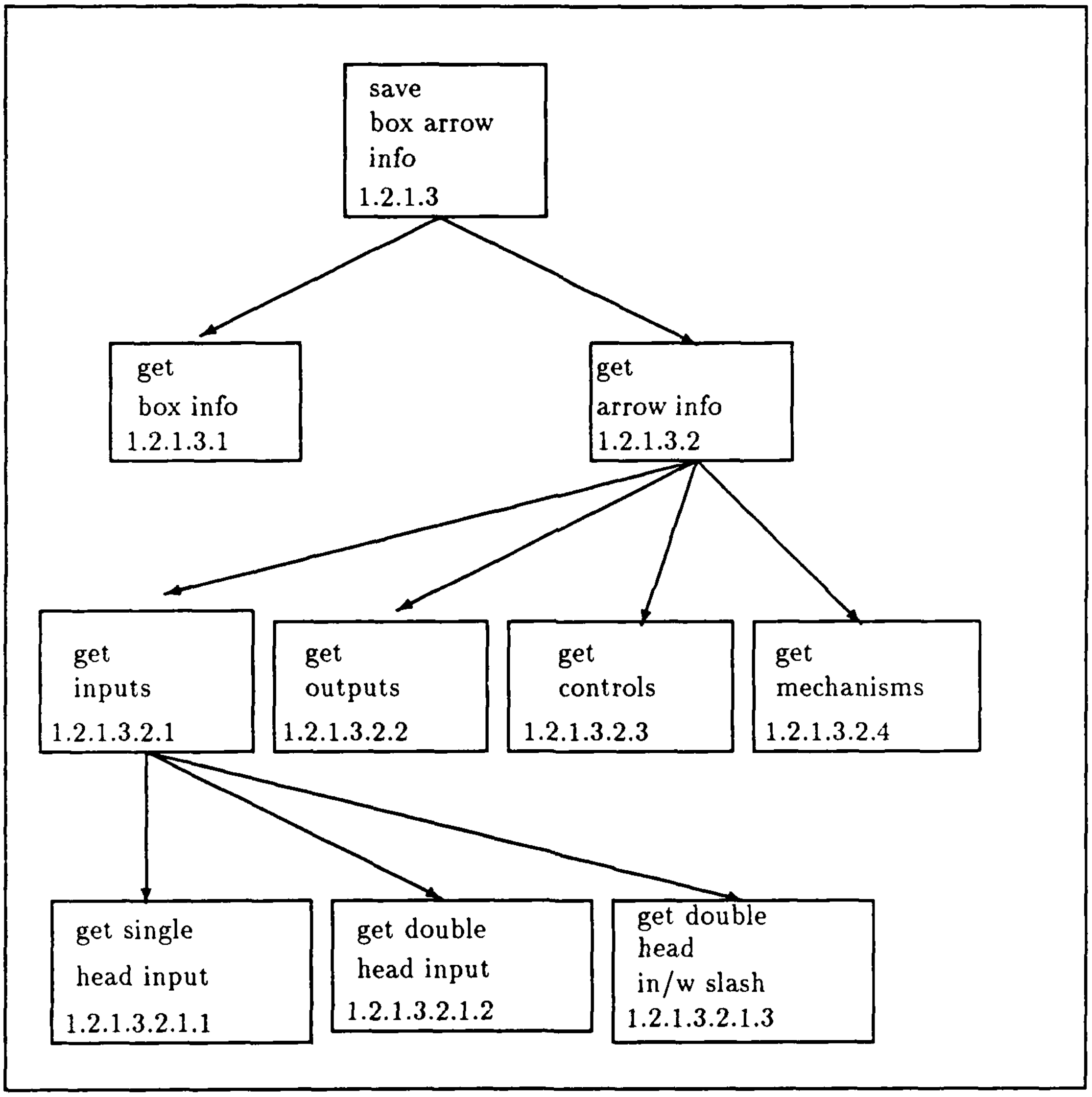


Figure B.2. Save Box Arrow Info(Module 1.2.1.3)

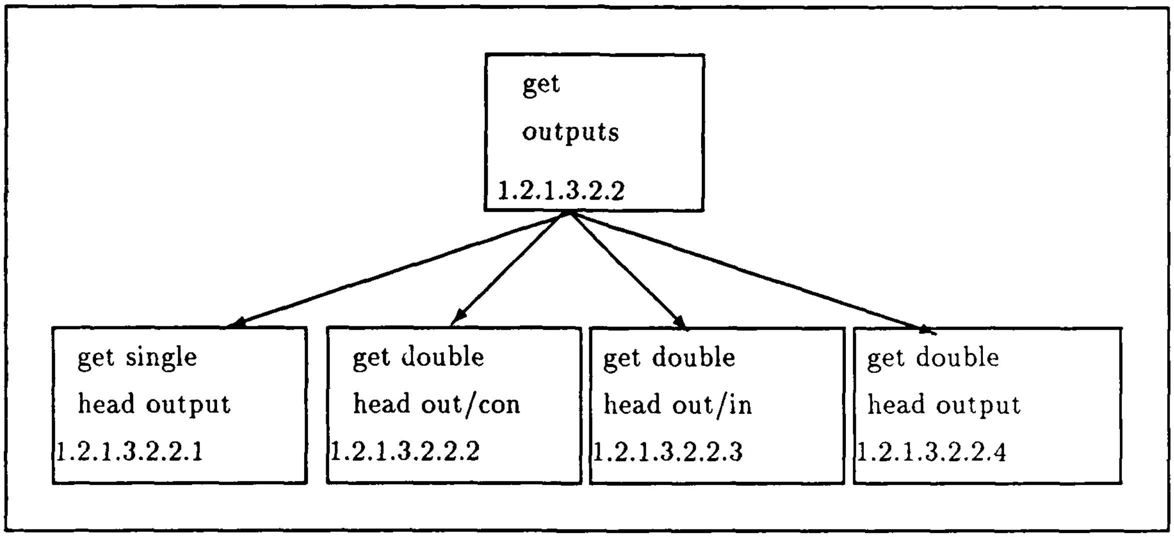


Figure B.3. Get Outputs(Module 1.2.1.3.2.2)

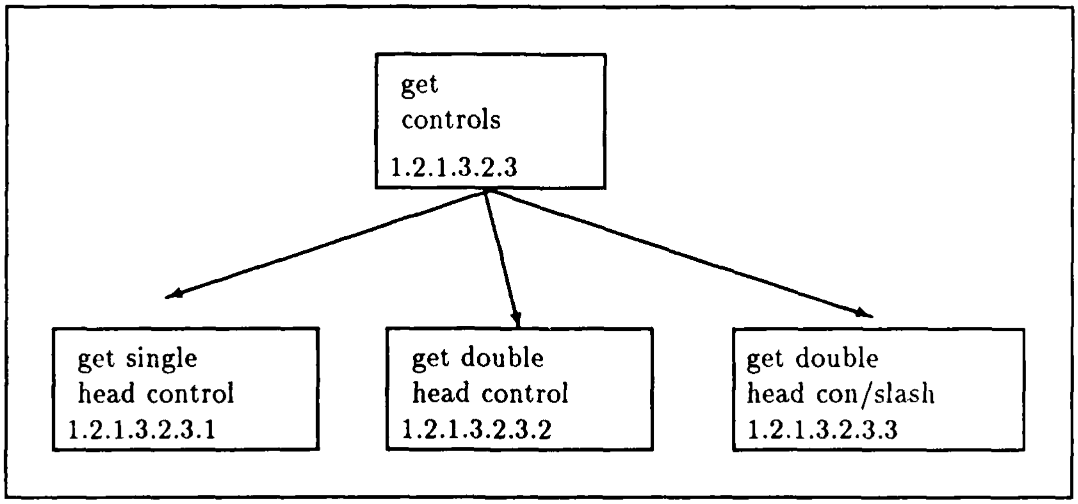


Figure B.4. Get Controls(Module 1.2.1.3.2.3)

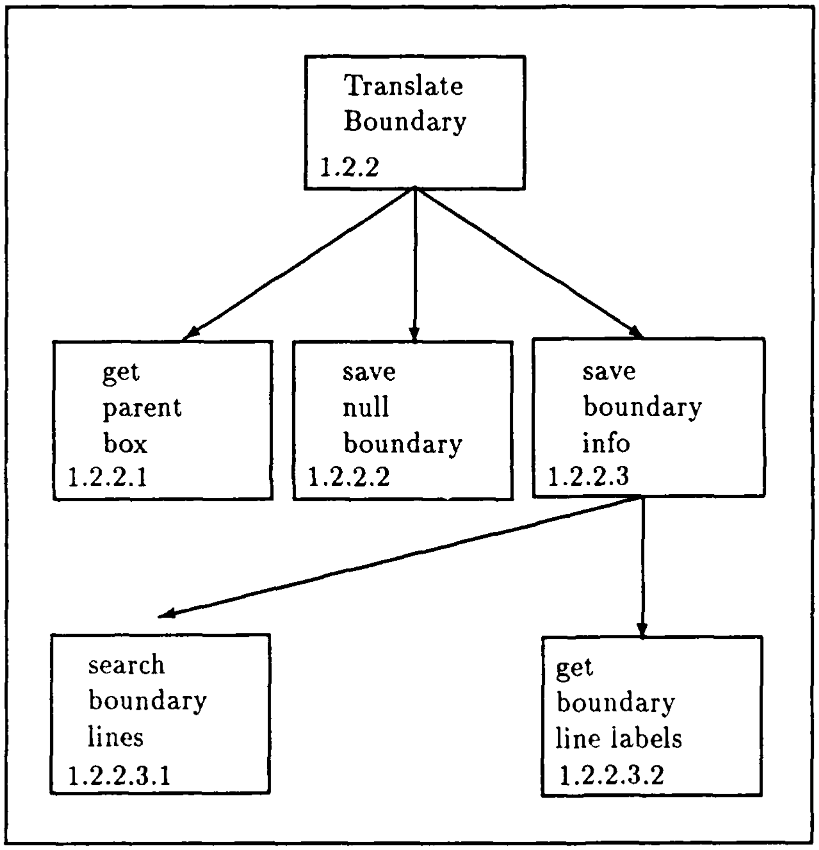


Figure B.5. Translate Boundary(Module 1.2.2)

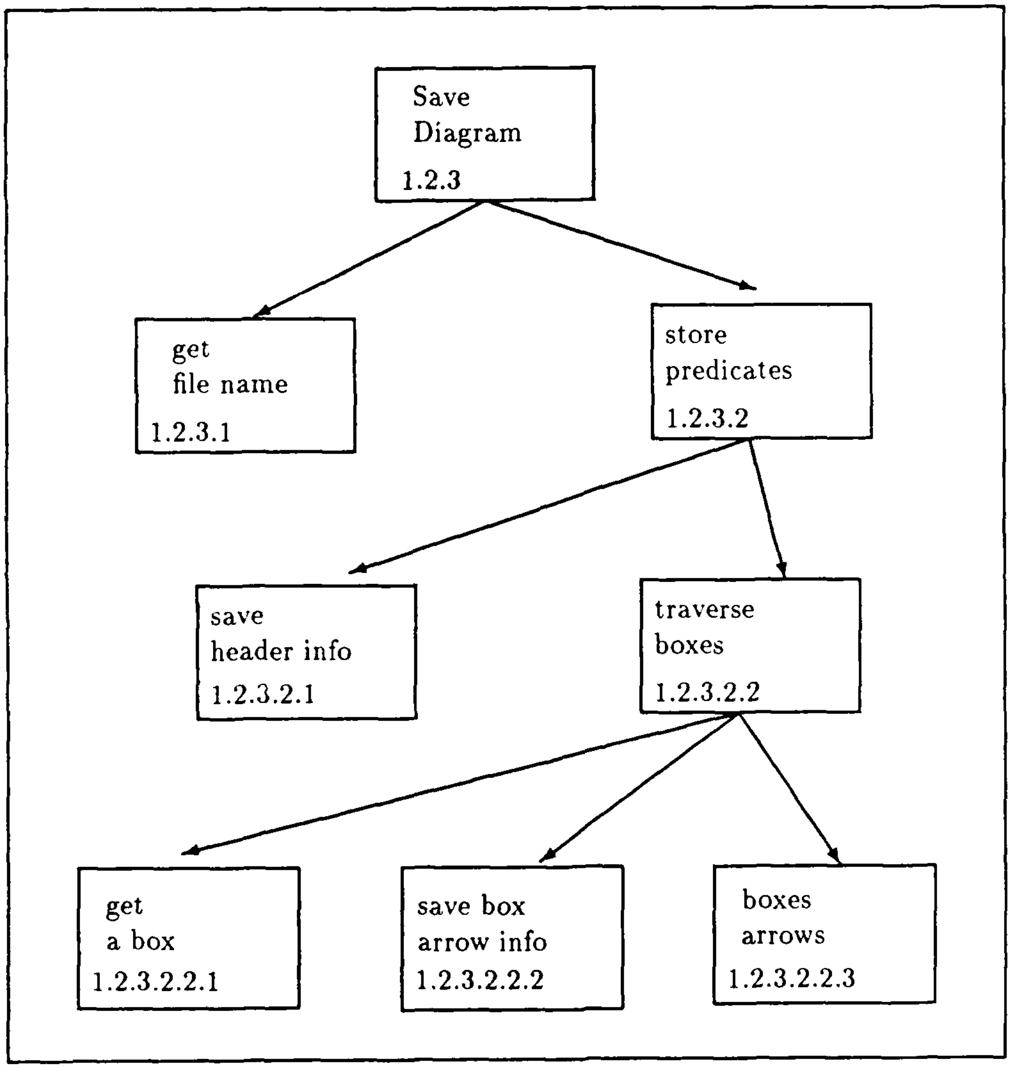


Figure B.6. Save Diagram(Module 1.2.3)

# Appendix C. *Data Structures of SAtool*

## *Introduction*

The purpose of this appendix is to discuss of the Data Structures of SAtool developed by Steven E. Johnson (9). A discussion of the data structures of the SAtool is needed because this thesis work should interface with the SAtool and use the $\mathrm{IDEF}_0$ diagrams and files generated by the SAtool. The SAtool allows users to interactively create and edit $\mathrm{IDEF}_0$ diagrams and to automatically produce the data dictionary information.

## *Data Structure*

Five primary data structures were designed to hold all the graphics and data dictionary information: the box structure, the line structure, the squiggle line structure, the header structure, and the footnote structure (9:4-11 - 4-14).

The box structure contains the information which is necessary to locate, name, and enumerate activity boxes (9:4-11). The activity boxes in the $\mathrm{IDEF}_0$ diagram are hooked by the linked list. The box structure uses two C pointers one for the next box structure and the other for pointing an activity data dictionary structure (9:4-11).

The line structure consists of the fields which are necessary to lacate, label, draw the lines, enumerate the lines to identify them, store the ICOM labels, and store the TO/FROM ALL labels (9:4-11 - 4-12). The line structure uses two numbers to identify the type of each end of the line (ie. arrowhead, tunnel, dot, turn right, or branch left, etc.) and uses C pointers to store the lines in binary trees with the root nodes (9:4-12). Figure C.1 shows four groups of the lines and the corresponding linked list structure is presented in Figure C.2 (9:4-12 - 4:13). The tree arrangement in Figure A.2 maps to how the line segments actually connect to one another and C pointer supports the simple recursive functions used to traverse the binary trees

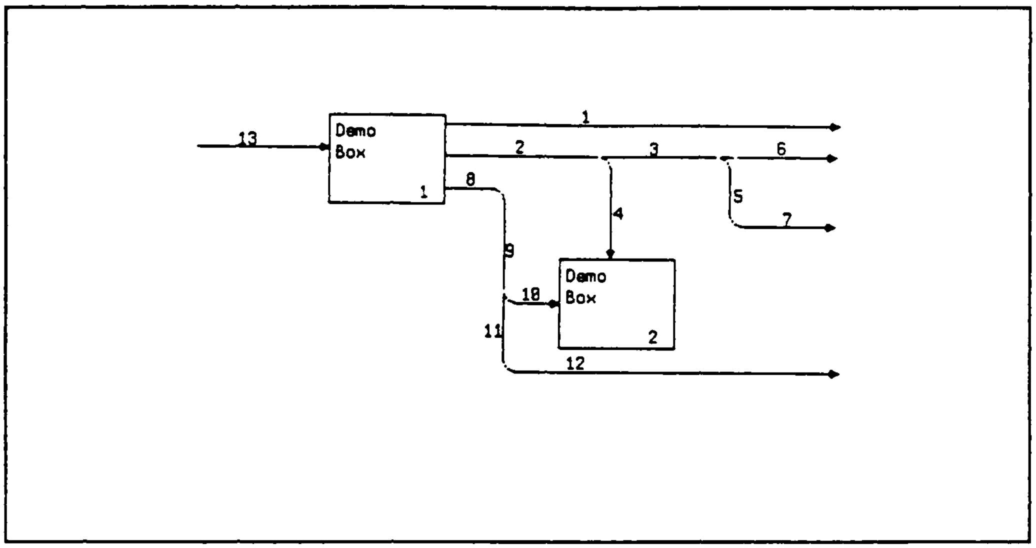


Figure C.1. Example Group of lines (9:4-12)

(9:4-12). The line structure uses another C pointer to point to a data divtionary information for a data element.

The squiggle line structure contains the information which is necessary to locate and to identify each squiggle line in the IDEF$_0$ diagram (9:4-13). The squiggle line structure uses a C pointer to store the squiggle lines for a particular IDEF$_0$ diagram in a single linked list (9:4-13 - 4-14).

The header structure consists of seven fields which are needed to draw, locate. and classify AUTHOR, DATE, PROJECT, REV, NODE, TITLE, and NUMBER of an IDEF$_0$ diagram(9:4-14 - 4-15). A single C pointer is used to save the header information since each IDEF$_0$ diagram only has one header (9:4-14).

Finally, the footnote structure contains the information which is needed to draw, locate and identify a matched pair of footnote labels (9:4-14). A C pointer is defined to point another footnote structure since the footnote structures for a IDEF$_0$ diagram are stored in a single linked list (9:4-14).

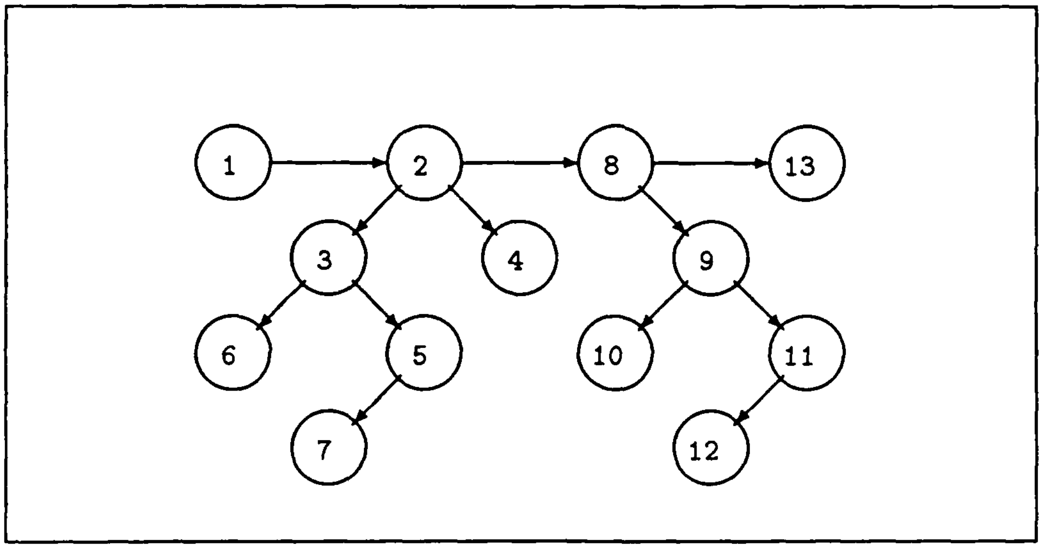


Figure C.2. The Linked List for Lines (9:4-13)

*Summary*

In this appendix, the data structure of the SAtool which is necessary to perform this thesis investigation was addressed from Johnson's effort. This information was used throughout this thesis effort.

# Appendix D. *User's Manual*

*User's Manual* introduces the basics of the ISES. The purpose of this manual provides a broad understanding of the ISES's operation, then provides a more detailed example for learning to use the ISES.

## *Table of Contents*

## *List of Figures*

Figure Page

*Introduction*

The $\text{IDEF}_0$ Syntax Expert System(ISES) is an interactive syntax check system. It provides a means for checking $\text{IDEF}_0$ syntax in any $\text{IDEF}_0$ diagrams drawn by SAtool and, producing error messages, error recovery, and editing suggestions. ISES allows the user to select a menu for drawing an $\text{IDEF}_0$ diagram and checking $\text{IDEF}_0$ syntax. The functions of the main menus include:

- *RECALL DGM* - read in a previously saved $\text{IDEF}_0$ diagram.
- *EDIT DGM* - edit an $\text{IDEF}_0$ diagram according to its attribute menus.
- *EDIT DD* - edit a data dictionary.
- *EDIT FPT* - edit a facing page text.
- *MISC FUNC* - save a diagram, change directory, exit SAtool, etc.
- *SAVE DGM* - save a graphics information and a data dictionary information.
- *CHECK SYNTAX* - check $\text{IDEF}_0$ syntax.

The first six menus are for drawing $\text{IDEF}_0$ diagrams, generating Data Dictionary information, and Facing Page Text and the last one is for ISES to check $\text{IDEF}_0$ syntax. A detailed guide for drawing $\text{IDEF}_0$ diagram is available in the user's manual of Johnson's thesis (9). This User's Manual focuses on the CHECK SYNTAX part. ISES runs on Sun3$^{TM}$ and Sun4$^{TM}$ workstations using the SunOS$^{TM}$ [1] and the SunView$^{TM}$ window-based environment. This manual explains how CHECK SYNTAX can be used to check $\text{IDEF}_0$ syntax. Some previous familiarity with $\text{IDEF}_0$ syntax and SAtool is required. Though not necessary, some familiarity with SunOS and SunView is helpful. Users should be thoroughly familiar with the concepts presented in this manual before using ISES.

[1] SunOS$^{TM}$ is a trademark of Sun Microsystems, Inc.

*The Mouse*

To move the cursor, place the mouse on its pad and move it in the desired direction. During the execution of SAtool, User is required to click the mouse button. All mouse button inputs should be clicked on the proper location in IDEF$_0$ diagram, otherwise, the mouse button inputs are ignored.

- *Right Button*

  The right button is used almost to abort operation of menu selected.

- *Middle Button*

  The middle button is used to exit SAtool (see Exit SAtool).

- *Left Button*

  The left mouse button is used to select one of menus and to start a menu operation.

*How to draw lines well*

Almost of the unnoticed errors are produced in the field of drawing lines. They provide a menas of generating unsuitable predicate data forms.

1. *Boundary lines*

   - All boundary lines should have their ICOM labels.
   - All arrows of Inputs, Mechanisms, and Controls must be touched on any box. The segment of lines inside an activity box is truncated automatically.
   - All output lines should be begun inside an activity box.

2. *Inter activity box lines*

   Every starting and ending point of the line segments should be begun and ended inside an activity box excepting the branch, join lines, and TO/FROM lines.

*Getting Started*

Set your UNIX path variable to include the ISES executable directory.

1. Enter *"suntools"*

   enter SunView and SunWindow environment.

2. Enter *"SAtool"*

   enter the IDEF$_0$ Diagram Translator environment.

3. The *Main Menu*

   Menus are displayed as the oval forms on the screen.

   Move the cursor to one the following choices to select:

   - *RECALL DGM*
   - *EDIT DGM*
   - *EDIT DD*
   - *EDIT FPT*
   - *MISC FUNC*
   - *SAVE DGM*
   - *CHECK SYNTAX*

4. *IDEF$_0$ diagram*

   By selecting one of the first five menus, The user is able to draw a new IDEF$_0$ diagram or update the previous IDEF$_0$ diagram.

5. *IDEF$_0$ Syntax*

   After drawing an IDEF$_0$ diagram, select *CHECK SYNTAX* oval by clicking the left mouse button on it. Now, three attribute submenus are displayed as the rectangular forms. Move the cursor to on. of the following choices to select:

   - *Activity*

- *Boundary*
- *Save(.pro)*

IDEF$_0$ syntax consists of *Activity* and *Boundary* IDEF$_0$ syntax.

6. *Activity IDEF$_0$ syntax*

   After clicking the left button on *Activity* rectangular box of the submcnus,

   (a) Move the cursor inside a box to be checked and click the left mouse button (Right - ABORT).

   (b) Enter the Prolog environment using another window.

7. *Boundary IDEF$_0$ Syntax*

   NOTE: *User must have the predicate file of the parent IDEF$_0$ diagram.*

   (a) After clicking the left button on *Boundary* rectangular box of menus,

   (b) move the cursor inside the input window and enter the file name with the parent activity box information (Right–ABORT).

   (c) Enter the Prolog environment using another window.

8. *Saving the predicate file*

   (a) After clicking the left mouse button on "save(.pro)" rectangular box of menus,

   (b) move the cursor inside the input window, and then enter the file name for the current IDEF$_0$ diagram. This file is a set of predicate data forms translated from the graphical information in the IDEF$_0$ diagram. It is used to check Boundary IDEF$_0$ syntax. The extension of the predicate file is a .pro.

9. *Prolog Environment*

   Enter *"prolog"*. This invokes the Prolog interpreter.

(a) Enter *"['ISES']."* - consult *ISES.*

Now, the following message is showed:

```
/***************************************************************/
/*                                                             */
/*          WELCOME TO IDEFO SYNTAX EXPERT SYSTEMS             */
/*                                                             */
/* I.   Type  start.    to begin a new session.                */
/* II.  Answer all questions using lower case and ending with*/
/*      a period.                                              */
/*                                                             */
/* III. Type     halt.    to exit Prolog session.              */
/*                                                             */
/***************************************************************/
```

(b) Enter *"start."* to begin checking IDEF$_0$ syntax of a clicked box in the current IDEF$_0$ diagram. Then, the message:

*Question: Do you want verbose operation(y./n.)?* is displayed. Enter *"y."* or *"n."*. In sace of *"y."*, the list of rules fired are shown and in case of *"n."*, only the IDEF$_0$ syntax messages are displayed. (See Examples).

(c) After then, the following message is shown on the screen:

```
Question: Do you wish to check ACTIVITY BOX, BOUNDARY ARROWS
  or to have HELP MESSAGES ?

  To check ACTIVITY BOX    -> Enter a.
  To check BOUNDARY ARROWS -> Enter b.
  To have HELP MESSAGES    -> Enter h.
Choice :
If checking Activity IDEF$_{0}$ syntax, enter "a.",
checking Boundary IDEF$_{0}$ syntax, enter "b.", or
wishing Help Message, enter "h.".
```

10. *IDEF$_0$ Syntax Message*

According to selection above description, the resulting messages of the IDEF$_0$ syntax checking procedure are displayed (see Examples).

11. *Trace*

The message, *Question: Do you wish to see how this answer was arrived at (y./n.)?* is followed by the resulting messages. Enter *"y."* or *"n."*. *"y."* means that the trace regarding the IDEF$_0$ syntax message derived is displayed and *"n."* skips (see Examples).

12. *Save Working Memory*

Then, the message, *Question: Do you wish to save the current working memory in a file (y./n.)?* is displayed. Enter *"y."* or *"n."*. In the case of *"y."*, the current working memory is saved in a file which is specified by the user and of *"n."*, the current working memory is erased automatically.

13. *Exit Prolog Environment*

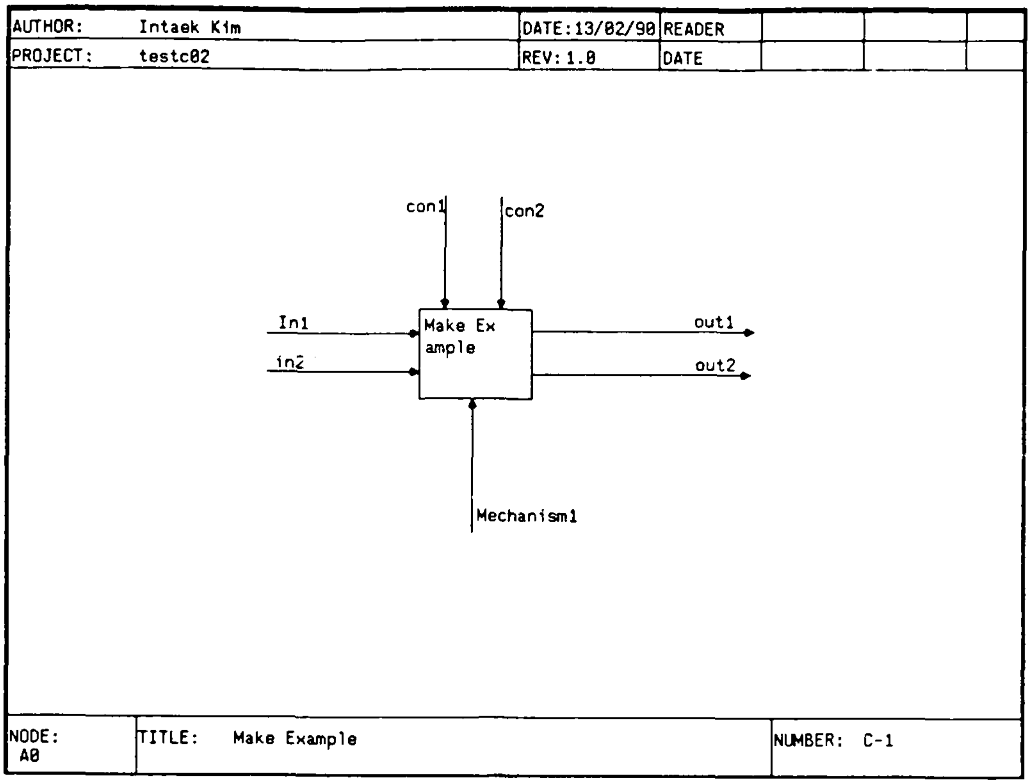
By entering "halt." or "ctrl c", prolog session is ended.

14. *Exit SAtool*

To exit SAtool, click the left mouse button on the "MISC FUNC" oval of the main menus. And then click the left mouse button on the "QUIT" under the "MISC FUNC" oval. Finally, click the middle button of the mouse.

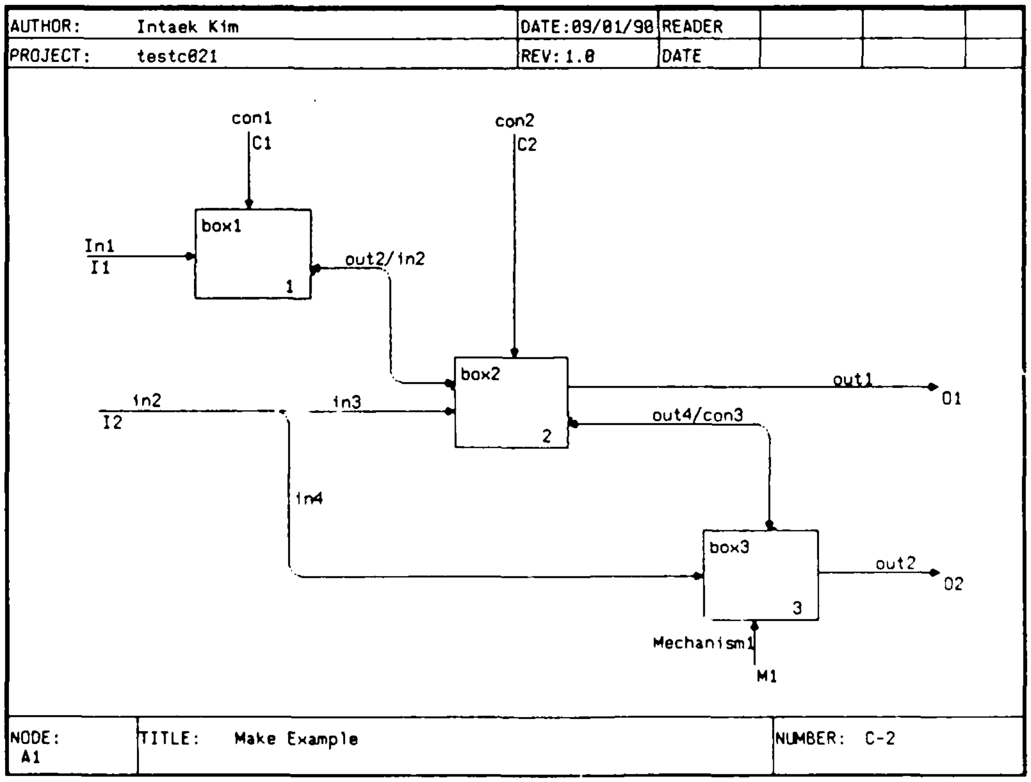
*Examples*

This section presents the actual demonstrations for checking process of the correct IDEF$_0$ diagram, however, the checking process about the IDEF$_0$ diagram with errors is the same as the correct case.



| | | | | | | |
|---|---|---|---|---|---|---|
| AUTHOR: | Intaek Kim | DATE:13/02/90 | READER | | | |
| PROJECT: | testc02 | REV: 1.0 | DATE | | | |

| | | |
|---|---|---|
| NODE: A0 | TITLE: Make Example | NUMBER: C-1 |



| | | | | | | |
|---|---|---|---|---|---|---|
| AUTHOR: | Intaek Kim | DATE:09/01/90 | READER | | | |
| PROJECT: | testc021 | REV: 1.0 | DATE | | | |

| | | |
|---|---|---|
| NODE: A1 | TITLE: Make Example | NUMBER: C-2 |

Figure D.1. Test IDFF$_0$ Diagram

```
 ares> prolog

Quintus Prolog Release 2.4.2 (Sun-4, SunOS 4.0)
Copyright (C) 1988, Quintus Computer Systems, Inc.  All rights reserved.
1310 Villa Street, Mountain View, California  (415) 965-7700

| ?- ['ISES'].
[consulting /usr2/eng/ikim/SAtoolExpert/ISES...]
   /**************************************************************/
   /*                                                            */
   /*          WELCOME TO IDEFO SYNTAX EXPERT SYSTEMS            */
   /*                                                            */
   /* I.  Type   start.    to begin a new session.               */
   /*                                                            */
   /* II. Answer all questions using lower case, ending with     */
   /*     a period.                                              */
   /*                                                            */
   /* III. Type    halt.    to exit prolog session.              */
   /*                                                            */
   /**************************************************************/

[ISES consulted 1.367 sec 19,008 bytes]

yes
| ?- start.
 Question: Do you want verbose operation(y./n.)? n.


 Question: Do you wish to check ACTIVITY BOX, BOUNDARY ARROWS
           or to have HELP MESSAGES ?

         To check ACTIVITY BOX    -> Enter a.
         To check BOUNDARY ARROWS -> Enter b.
         To have HELP MESSAGE     -> Enter h.
Choice : a.


/*************** IDEFO Syntax Messages ***************/

Name        --> CORRECT: Activity Name is OK.

Input       --> CORRECT: Input is OK.

Output      --> CORRECT: Output is OK.
```

```
Control      --> CORRECT: Control is OK.

Mechanism    --> CORRECT: Mechanism is OK.

Number       --> CORRECT: Activity number is OK.
                          This activity must be the top most level box.


 Question: Do you wish to see how this answer
           was arrived at(y./n.)? n.

     /*****************!!! WARNING !!!************************/
     /* After this session, all working memory elements will */
     /* be erased except for elements being protected by     */
     /*   keep   statements in the knowledge base.           */
     /********************************************************/

 Question: Do you wish to save the current working memory
in a file(y./n.)? n.
```

```
/*************************************************************/
/*                                                           */
/*          WELCOME TO IDEFO SYNTAX EXPERT SYSTEMS           */
/*                                                           */
/* I.  Type   start.    to begin a new session.              */
/*                                                           */
/* II. Answer all questions using lower case, ending with    */
/*     a period.                                             */
/*                                                           */
/* III. Type    halt.    to exit prolog session.             */
/*                                                           */
/*************************************************************/


yes
| ?- start.
 Question: Do you want verbose operation(y./n.)? y.


 Question: Do you wish to check ACTIVITY BOX, BOUNDARY ARROWS
           or to have HELP MESSAGES ?

         To check ACTIVITY BOX    -> Enter a.
         To check BOUNDARY ARROWS -> Enter b.
         To have HELP MESSAGE     -> Enter h.
Choice : a.



  /*************** IDEFO Syntax Messages ***************/
Trying rule1:: [Name, ,     --> ERROR: No Activity Name.
                     Each box must have an activity name]
Trying rule2:: [Name, ,     --> CORRECT: Activity Name is OK]
Proved rule2:: [Name, ,     --> CORRECT: Activity Name is OK]

Name        --> CORRECT: Activity Name is OK.
Trying rule3:: [Input, ,    --> CORRECT: No Input Arrows, however,
                         Input is OK]
Trying rule4:: [Input, ,    --> ERROR: No Input Label
                   Each Input arrow must have an input label]
Trying rule5:: [Input, ,    --> RECOMMEND:
                   You would better reduce the number of Input arrows
                   from 0 to 5]
Trying rule6:: [Input, ,    --> CORRECT: Input is OK]
Proved rule6:: [Input, ,    --> CORRECT: Input is OK]
```

```
Input       --> CORRECT: Input is OK.
Trying rule7:: [Output, ,   --> ERROR: You should have at least
                       one output arrow]
Trying rule8:: [Output, ,   --> ERROR: No Output Label.
                     Each Output Arrow should have an output Label]
Trying rule9:: [Output, ,   --> RECOMMEND:
               You would better reduce the number of Output arrows
                          from 1 to 5]
Trying rule10:: [Output, ,   --> CORRECT: Output is OK]
Proved rule10:: [Output, ,   --> CORRECT: Output is OK]

Output      --> CORRECT: Output is OK.
Trying rule11:: [Control, ,  --> ERROR: You should have at least
                one control arrow]
Trying rule12:: [Control, ,  --> ERROR: No Control Label.
                 Each Control Arrow should have a control Label]
Trying rule13:: [Control, ,  --> RECOMMEND:
                 You would better reduce the number of Control arrows
                      from 1 to 5]
Trying rule14:: [Control, ,  --> CORRECT: Control is OK]
Proved rule14:: [Control, ,  --> CORRECT: Control is OK]

Control     --> CORRECT: Control is OK.
Trying rule15:: [Mechanism, ,--> ERROR: No Mechanism Label.
                 Each Mechanism Arrow should have a mechanism Label]
Trying rule16:: [Mechanism, ,--> CORRECT: No Mechanism Arrows, however,
                       Mechanism is OK]
Trying rule17:: [Mechanism, ,--> RECOMMEND:
                You would better reduce the number of Mechanism arrows
                      from 0 to 5]
Trying rule18:: [Mechanism, ,--> CORRECT: Mechanism is OK]
Proved rule18:: [Mechanism, ,--> CORRECT: Mechanism is OK]

Mechanism   --> CORRECT: Mechanism is OK.
Trying rule19:: [Number, ,   --> CORRECT: Activity number is OK.
                        This activity must be the top most level box]
Proved rule19:: [Number, ,   --> CORRECT: Activity number is OK.
                        This activity must be the top most level box]

Number      --> CORRECT: Activity number is OK.
                        This activity must be the top most level box.
```

```
 Question: Do you wish to see how this answer
          was arrived at(y./n.)? y.
GOAL::   [Name, ,     --> CORRECT: Activity Name is OK]
GOAL::   [Input, ,    --> CORRECT: Input is OK]
GOAL::   [Output, ,   --> CORRECT: Output is OK]
GOAL::   [Control, ,  --> CORRECT: Control is OK]
GOAL::   [Mechanism, ,--> CORRECT: Mechanism is OK]
GOAL::   [Number, ,   --> CORRECT: Activity number is OK.
                         This activity must be the top most level box]
rule19:: [Number, ,   --> CORRECT: Activity number is OK.
          This activity must be the top most level box]  Was Derived From
        [title,is,Make Example] AND
        [Make Example,number_is,0] AND
        [0,==,0]
SOLVED:: [0,==,0]
TOLD::   [Make Example,number_is,0]
TOLD::   [title,is,Make Example]
rule18:: [Mechanism, ,--> CORRECT: Mechanism is OK]  Was Derived From
        [activityname,is,Make Example]
KNOWN::  was_told:: [activityname,is,Make Example]
rule14:: [Control, ,  --> CORRECT: Control is OK]  Was Derived From
        [activityname,is,Make Example]
KNOWN::  was_told:: [activityname,is,Make Example]
rule10:: [Output, ,   --> CORRECT: Output is OK]  Was Derived From
        [activityname,is,Make Example]
KNOWN::  was_told:: [activityname,is,Make Example]
rule6:: [Input, ,    --> CORRECT: Input is OK]  Was Derived From
        [activityname,is,Make Example]
KNOWN::  was_told:: [activityname,is,Make Example]
rule2:: [Name, ,     --> CORRECT: Activity Name is OK]  Was Derived From
        [activityname,is,Make Example] AND
        [Make Example,\==,]
SOLVED:: [Make Example,\==,]
TOLD::   [activityname,is,Make Example]


     /*****************!!! WARNING !!!************************/
     /* After this session, all working memory elements will */
     /* be erased except for elements being protected by     */
     /*   keep  statements in the knowledge base.            */
     /********************************************************/

 Question: Do you wish to save the current working memory
in a file(y./n.)? y.
```

```
Please supply a filename: ’example.wm’.
```

```
/**************************************************************/
/*                                                            */
/*          WELCOME TO IDEFO SYNTAX EXPERT SYSTEMS            */
/*                                                            */
/* I. Type   start.    to begin a new session.               */
/*                                                            */
/* II. Answer all questions using lower case, ending with     */
/*     a period.                                              */
/*                                                            */
/* III. Type    halt.    to exit prolog session.              */
/*                                                            */
/**************************************************************/


yes
| ?- start.
 Question: Do you want verbose operation(y./n.)? n.


 Question: Do you wish to check ACTIVITY BOX, BOUNDARY ARROWS
           or to have HELP MESSAGES ?

         To check ACTIVITY BOX    -> Enter a.
         To check BOUNDARY ARROWS -> Enter b.
         To have HELP MESSAGE     -> Enter h.
Choice : b.



  /*************** IDEFO Syntax Messages ***************/

Boundary Input       --> CORRECT: Boundary Input is OK.

Boundary Output      --> CORRECT:
                         Boundary Output is OK.

Boundary Control     --> CORRECT: Boundary
                         Control is OK.

Boundary Mechanism   --> CORRECT: Boundary
                         Mechanism is OK.


 Question: Do you wish to see how this answer
```

```
           was arrived at(y./n.)? n.

     /*****************!!! WARNING !!!************************/
     /* After this session, all working memory elements will */
     /* be erased except for elements being protected by     */
     /*   keep   statements in the knowledge base.           */
     /********************************************************/

 Question: Do you wish to save the current working memory
in a file(y./n.)? n.
```

```
/***************************************************************/
/*                                                             */
/*          WELCOME TO IDEFO SYNTAX EXPERT SYSTEMS             */
/*                                                             */
/* I.  Type   start.    to begin a new session.                */
/*                                                             */
/* II. Answer all questions using lower case, ending with      */
/*     a period.                                               */
/*                                                             */
/* III. Type    halt.    to exit prolog session.               */
/*                                                             */
/***************************************************************/


yes
| ?- start.
 Question: Do you want verbose operation(y./n.)? y.


 Question: Do you wish to check ACTIVITY BOX, BOUNDARY ARROWS
           or to have HELP MESSAGES ?

         To check ACTIVITY BOX    -> Enter a.
         To check BOUNDARY ARROWS -> Enter b.
         To have HELP MESSAGE     -> Enter h.
Choice : b.



  /*************** IDEFO Syntax Messages ***************/
Trying rule2:: [Boundary Input, ,  --> !!! THIS IS A FATAL ERROR !!!]
Trying rule1:: [boundarysarule,is,stalled]
Trying rule6:: [Boundary Input, , --> ERROR: No boundary input label]
Trying rule7:: [Boundary Input, ,--> ERROR: Parent Input has no label]
Trying rule8:: [Boundary Input, ,--> ERROR: The number of Input
         arrow(s) of Parent Activity box is greater than that of
         Boundary Input arrow(s) -- Must have the same number]
Trying rule9:: [Boundary Input, ,    --> ERROR: The number of Input
         arrow(s) of Parent Activity box is less than that of
         Boundary Input arrow(s) -- Must have the same number]
Trying rule10:: [Boundary Input, ,    --> RECOMMEND:
         You would better reduce the number of arrows to six
         below]
Trying rule11:: [Boundary Input, ,    --> CORRECT: Boundary Input is OK]
Trying rule13:: [Boundary Input, ,    --> CORRECT: Boundary Input is OK]
Trying rule12:: [case_of_boundary_in,is,1]
Trying rule14:: [Boundary Input, ,    --> CORRECT: Boundary Input is OK]
Trying rule12:: [case_of_boundary_in,is,2]
Proved rule12:: [case_of_boundary_in,is,2]
Proved rule14:: [Boundary Input, ,    --> CORRECT: Boundary Input is OK]

Boundary Input       --> CORRECT: Boundary Input is OK.
Trying rule3:: [Boundary Output, ,   -->
There is nothing about Parent activity]
Trying rule1:: [boundarysarule,is,stalled]
Trying rule19:: [Boundary Output, ,--> ERROR: No boundary output label]
Trying rule20:: [Boundary Output, ,   --> ERROR:
 Parent Output has no label]
Trying rule21:: [Boundary Output, ,--> ERROR: No boundary output arrow.
                         Should have at least one boundary output arrow]
Trying rule22:: [Boundary Output, ,   --> ERROR: No parent output arrow.
                        Should have at least one parent output arrow]
Trying rule23:: [Boundary Output, ,   --> ERROR: The number of Output
         arrow(s) of Parent Activity box is greater than that of
         Boundary Output arrow(s) -- Must have the same number]
Trying rule24:: [Boundary Output, ,   --> ERROR: The number of Output
         arrow(s) of Parent Activity box is less than that of
         Boundary Output arrow(s) -- Must have the same number]
Trying rule25:: [Boundary Output, ,    --> RECOMMEND:
         You would better reduce the number of arrows to six
         below]
Trying rule27:: [Boundary Output, ,   --> CORRECT:
                         Boundary Output is OK]
Trying rule26:: [case_of_boundary_out,is,1]
Trying rule28:: [Boundary Output, ,   --> CORRECT:
                         Boundary Output is OK]
Trying rule26:: [case_of_boundary_out,is,2]
Proved rule26:: [case_of_boundary_out,is,2]
Proved rule28:: [Boundary Output, ,   --> CORRECT:
                         Boundary Output is OK]

Boundary Output      --> CORRECT:
                         Boundary Output is OK.
Trying rule4:: [Boundary Control, ,  -->
Maybe you have tried to check syntax with
                a file without PARENT ACTIVITY BOX information]
Trying rule1:: [boundarysarule,is,stalled]
Trying rule33:: [Boundary Control, ,--> ERROR: No boundary control label]
Trying rule34:: [Boundary Control, ,--> ERROR: Parent Control has no label]
Trying rule35:: [Boundary Control, ,  --> ERROR: No boundary control arrow.
```

```
                         Should have at least one boundary control arrow]
Trying rule36:: [Boundary Control, ,  --> ERROR: No parent control arrow.
                        Should have at least one parent control arrow]
Trying rule37:: [Boundary Control, ,  --> RECOMMEND:
         You would better reduce the number of arrows to six
         below]
Trying rule39:: [Boundary Control, ,  --> CORRECT: Boundary
                         Control is OK]
Trying rule38:: [case_of_boundary_con,is,1]
Trying rule40:: [Boundary Control, ,  --> CORRECT: Boundary
                         Control is OK]
Trying rule38:: [case_of_boundary_con,is,2]
Proved rule38:: [case_of_boundary_con,is,2]
Proved rule40:: [Boundary Control, ,  --> CORRECT: Boundary
                         Control is OK]

Boundary Control     --> CORRECT: Boundary Control is OK.
Trying rule5:: [Boundary Mechanism, ,--> PLEASE START AGAIN !!!]
Trying rule1:: [boundarysarule,is,stalled]
Trying rule45:: [Boundary Mechanism, ,--> ERROR:
 No boundary mechanism label]
Trying rule46:: [Boundary Mechanism, ,--> ERROR:
 Parent Mechanism has no label]
Trying rule47:: [Boundary Mechanism, ,--> ERROR: The number of
         Mechanism arrow(s) of Parent Activity box is greater than that
         of Boundary Mechanism arrow(s) -- Must have the same
         number]
Trying rule48:: [Boundary Mechanism, ,--> ERROR: The number of
         Mechanism arrow(s) of Parent Activity box is less than
         that of Boundary Mechanism arrow(s) -- Must have the same
         number]
Trying rule49:: [Boundary Mechanism, ,--> RECOMMEND:
         You would better reduce the number of arrows to six
         below]
Trying rule50:: [Boundary Mechanism, ,--> CORRECT: Boundary
                         Mechanism is OK]
Trying rule52:: [Boundary Mechanism, ,--> CORRECT: Boundary
                         Mechanism is OK]
Trying rule51:: [case_of_boundary_mech,is,1]
Proved rule51:: [case_of_boundary_mech,is,1]
Proved rule52:: [Boundary Mechanism, ,--> CORRECT: Boundary
                         Mechanism is OK]

Boundary Mechanism   --> CORRECT: Boundary
                         Mechanism is OK.
 Question: Do you wish to see how this answer
           was arrived at(y./n.)? y.
GOAL::   [Boundary Input, ,    --> CORRECT: Boundary Input is OK]
GOAL::   [Boundary Output, ,   --> CORRECT:
                         Boundary Output is OK]
GOAL::   [Boundary Control, ,  --> CORRECT: Boundary
                         Control is OK]
GOAL::   [Boundary Mechanism, ,--> CORRECT: Boundary
                         Mechanism is OK]
rule52:: [Boundary Mechanism, ,--> CORRECT: Boundary
                         Mechanism is OK]  Was Derived From
        [case_of_boundary_mech,is,1] AND
        [child_title,is,Make Example] AND
        [Make Example,mechanism_is,Mechanism1] AND
        [boundary_mechanism1,is,Mechanism1]
TOLD::   [boundary_mechanism1,is,Mechanism1]
TOLD::   [Make Example,mechanism_is,Mechanism1]
KNOWN::  was_told:: [child_title,is,Make Example]
rule51:: [case_of_boundary_mech,is,1]  Was Derived From
        [boundary_mechanism,has_number,1] AND
        [child_title,is,Make Example] AND
        [Make Example,has_mechanism_number,1]
TOLD::   [Make Example,has_mechanism_number,1]
KNOWN::  was_told:: [child_title,is,Make Example]
TOLD::   [boundary_mechanism,has_number,1]
rule40:: [Boundary Control, ,  --> CORRECT: Boundary
                         Control is OK]  Was Derived From
        [case_of_boundary_con,is,2] AND
        [boundary_control1,is,con1] AND
        [boundary_control2,is,con2] AND
        [child_title,is,Make Example] AND
        [Make Example,control_is,con1] AND
        [Make Example,control_is,con2]
TOLD::   [Make Example,control_is,con2]
TOLD::   [Make Example,control_is,con1]
KNOWN::  was_told:: [child_title,is,Make Example]
TOLD::   [boundary_control2,is,con2]
TOLD::   [boundary_control1,is,con1]
rule38:: [case_of_boundary_con,is,2]  Was Derived From
        [boundary_control,has_number,2] AND
        [child_title,is,Make Example] AND
        [Make Example,has_control_number,2]
```

```
TOLD::   [Make Example,has_control_number,2]
KNOWN::  was_told:: [child_title,is,Make Example]
TOLD::   [boundary_control,has_number,2]
rule28:: [Boundary Output, ,   --> CORRECT:
                         Boundary Output is OK]  Was Derived From
        [case_of_boundary_out,is,2] AND
        [boundary_output1,is,out1] AND
        [boundary_output2,is,out2] AND
        [child_title,is,Make Example] AND
        [Make Example,output_is,out1] AND
        [Make Example,output_is,out2]
TOLD::   [Make Example,output_is,out2]
TOLD::   [Make Example,output_is,out1]
KNOWN::  was_told:: [child_title,is,Make Example]
TOLD::   [boundary_output2,is,out2]
TOLD::   [boundary_output1,is,out1]
rule26:: [case_of_boundary_out,is,2]  Was Derived From
        [boundary_output,has_number,2] AND
        [child_title,is,Make Example] AND
        [Make Example,has_output_number,2]
TOLD::   [Make Example,has_output_number,2]
KNOWN::  was_told:: [child_title,is,Make Example]
TOLD::   [boundary_output,has_number,2]
rule14:: [Boundary Input, ,    --> CORRECT: Boundary Input is OK]
    Was Derived From
        [case_of_boundary_in,is,2] AND
        [boundary_input1,is,in2] AND
        [boundary_input2,is,In1] AND
        [child_title,is,Make Example] AND
        [Make Example,input_is,in2] AND
        [Make Example,input_is,In1]
TOLD::   [Make Example,input_is,In1]
TOLD::   [Make Example,input_is,in2]
KNOWN::  was_told:: [child_title,is,Make Example]
TOLD::   [boundary_input2,is,In1]
TOLD::   [boundary_input1,is,in2]
rule12:: [case_of_boundary_in,is,2]  Was Derived From
        [boundary_input,has_number,2] AND
        [child_title,is,Make Example] AND
        [Make Example,has_input_number,2]
TOLD::   [Make Example,has_input_number,2]
TOLD::   [child_title,is,Make Example]
TOLD::   [boundary_input,has_number,2]
```

```
     /*****************!!! WARNING !!!************************/
     /* After this session, all working memory elements will */
     /* be erased except for elements being protected by     */
     /*   keep   statements in the knowledge base.           */
     /********************************************************/

 Question: Do you wish to save the current working memory
in a file(y./n.)? n.
```

```
/**************************************************************/
/*                                                            */
/*          WELCOME TO IDEFO SYNTAX EXPERT SYSTEMS           */
/*                                                            */
/* I.  Type   start.    to begin a new session.               */
/*                                                            */
/* II. Answer all questions using lower case, ending with    */
/*     a period.                                              */
/*                                                            */
/* III. Type    halt.    to exit prolog session.              */
/*                                                            */
/**************************************************************/


yes
| ?- halt.
ares>
```

# Appendix E. *Programmer's Guide*

*Programmer's Guide* introduces several topics of interest to ISES programmers and developers.

## *Table of Contents*

## *List of Figures*

Figure Page

### *Introduction*

The focus of this thesis effort was to design and implement an application of expert system formulation for checking $\mathrm{IDEF}_0$ syntax of $\mathrm{IDEF}_0$ diagrams as derived from SAtool. The work in this thesis is divided into two major ctegories: $\mathrm{IDEF}_0$ Diagram Translator and $\mathrm{IDEF}_0$ Syntax Expert System. The $\mathrm{IDEF}_0$ Diagram Translator translates the $\mathrm{IDEF}_0$ diagrams into a set of predicate forms and the predicate forms file is used as the data base of the $\mathrm{IDEF}_0$ Syntax Expert System. The $\mathrm{IDEF}_0$ Syntax Expert System checks the $\mathrm{IDEF}_0$ syntax of $\mathrm{IDEF}_0$ diagrams. The objective of this appendix is to specify the procedure for generating the executable files and to outline some basic concepts of the translator and expert system.

### *Software Documentation*

The existing source code is fully documented in AFIT System Development Documentation Guidelines and Standards (8). The following list shows the file header of the source codes.

- *DATE:* Date of current version number.
- *VERSION:* Current version number.
- *TITLE:* Title for this file.
- *FILENAME:* File name for the module.
- *DESCRIPTION:* Description of the module's function.
- *AUTHOR:* Name of one who responsible for this file.
- *PROJECT:* Name of the software project of which this file is a part.
- *OPERATING SYSTEM:* Name and version number of operating system under which this file was written.
- *LANGUAGE:* Name of language used for source code.
- *FILE PROCESSING:* How the file is used.

- *CONTENTS:* Modules contained in the file.
- *HISTOKY:* List of major changes to the file.

The following list presents the subroutine header of the source codes.

- *DATE:* Date of the module.
- *VERSION:* Current version number.
- *NAME:* Module name.
- *MODULE NUMBER:* Module number of current program.
- *DESCRIPTION:* Text description of the module's function.
- *ALGORITHM:* Algorithm used.
- *PASSED VARIABLES:* Variables passed to the module.
- *RETURNS:* Value returned by the module.
- *GLOBAL VARIABLES USED:* Variables read by the module.
- *GLOBAL VARIABLES CHANGED:* Variables changed by the module.
- *FILES READ:* Files read by the module.
- *FILES WRITTEN:* Files written by the module.
- *HARDWARE INPUT:* I/O ports read.
- *HARDWARE OUTPUT:* I/O ports read.
- *MODULES CALLED:* Other procedures called.
- *CALLING MODULES:* What modules call.
- *AUTHOR:* One who wrote this module.
- *HISTORY:* List of major changes to the module.

```
OBJECTS = main.o dataddict.o messages.o boxfunctions.o
          headerfunctions.o editboxfunc.o
          miscfunctions.o ddsearchfuncs.o
          endfuncs.o find.o morelinefuncs.o
          linelabel.o moreddfuncs.o ddsearchfuncs.o
          savefuncs.o
          fptfuncs.o sqglefuncs.o fnotefuncs.o
          moresave.o screendump.o readfuncs.o
          session.o syntaxfuncs.o
HEADERS = globals.h
ALL = sad
CFLAGS = -O
LIBS = -lsuntool -lsunwindow -lpixrect -lm
sad : $(OBJECTS)
cc $(CFLAGS) $(OBJECTS) $(LIBS) -o SAtool
```

Figure E.1. Contents of Makefile

*Make File*

The file of the IDEF$_0$ Diagram Translator is included in the files of the SAtool because the IDEF$_0$ Diagram Translator was coded as a part of the SAtool under the SunOS$^{TM}$. The file name of source code for the IDEF$_0$ Diagram Translator is *syntaxfuncs.c.* The executable file was produced by using the UNIX *make* utility. Figure E.1 shows the contents of the *Makefile* file. The system command *make* causes to be compiled and linked all together and generated the executable file, *SAtool.*

*Files produced by IDT*

**.pro* This file contains a set of predicate data forms into which the IDEF$_0$ Diagram Translator translates the IDEF$_0$ diagram. The symbol * is a file name which the user specifies. The extension of the file is added automatically. This file is used to check the IDEF$_0$ syntax of boundary arrows in any IDEF$_0$ diagram. Figure E.3 shows an example of the predicate data file translated from the above IDEF$_0$

diagram shown in Figure E.2. Also, Figure E.4 shows an example of the predicate data file from the below IDEF$_0$ diagram shown in Digure E.2.

*CHECKBOX.PRO* This file is a temporary file which is created and overwrited automatically. Also this file is used for checking the IDEF$_0$ syntax of an activity box which user specifies in any IDEF$_0$ diagram. Figure E.5 represents the predicate data file of the activity box, *box1*, which is clicked by the user in Figure E.2.

*CHECKBOUNDARY.PRO* This file contains the predicate data forms of the boundary arrows and its parent activity box and is a temporary file which is created and overwrited automatically. Also this file becomes the data base (working memory) of the IDEF$_0$ Syntax Expert System. Figure E.6 shows the predicate data file of boundary arrows in Figure E.2.

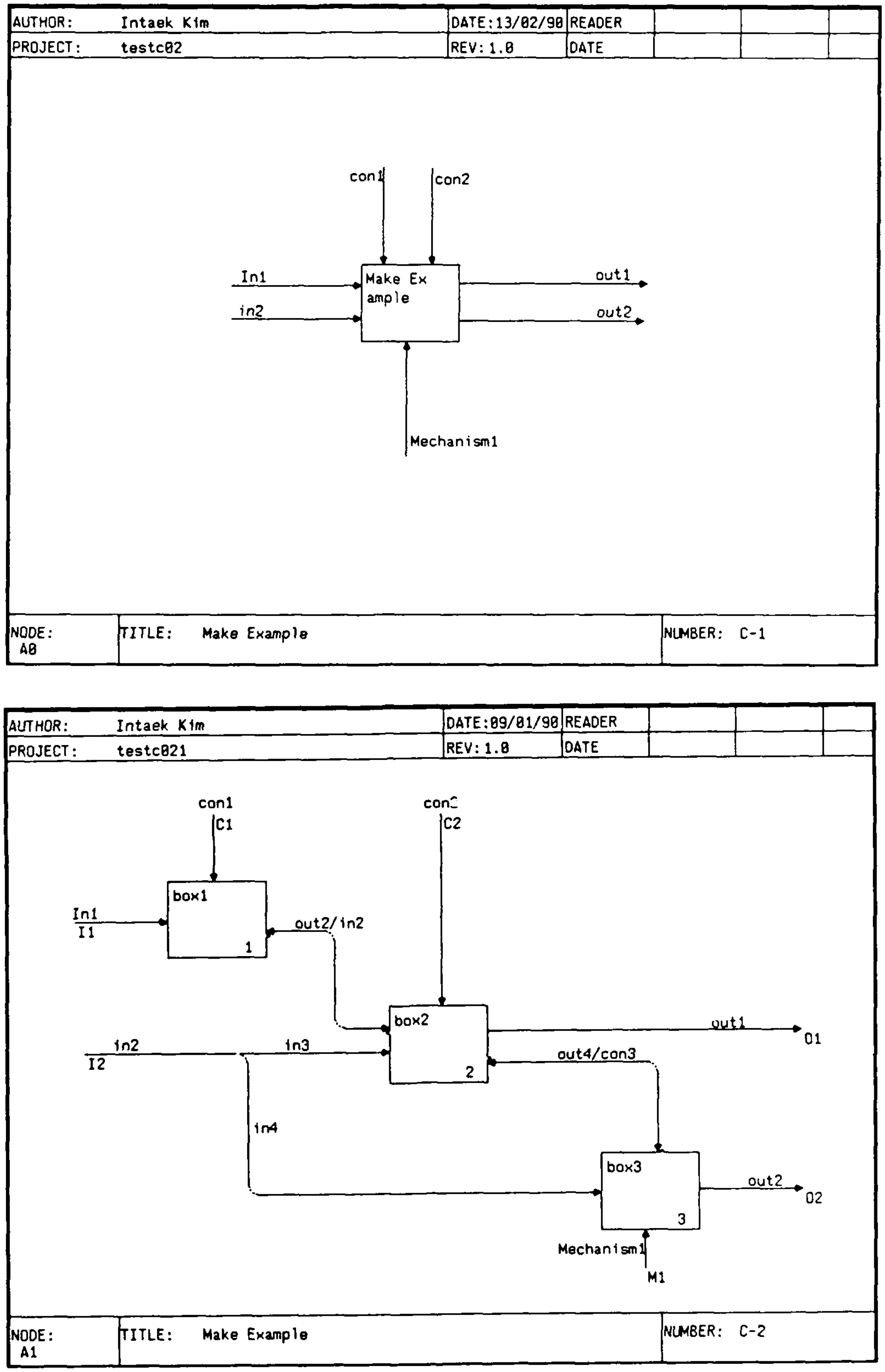
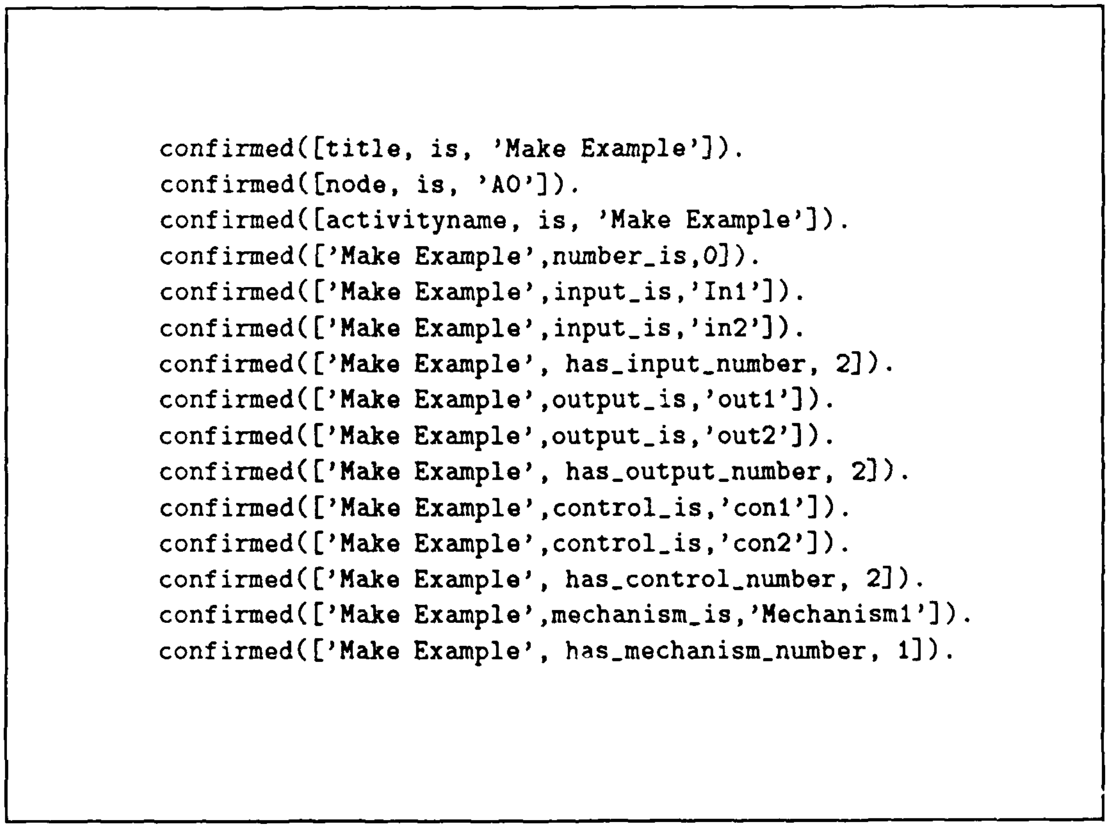


Figure E.2. Example of $IDEF_0$ Diagram



```
confirmed([title, is, 'Make Example']).
confirmed([node, is, 'A0']).
confirmed([activityname, is, 'Make Example']).
confirmed(['Make Example',number_is,0]).
confirmed(['Make Example',input_is,'In1']).
confirmed(['Make Example',input_is,'in2']).
confirmed(['Make Example', has_input_number, 2]).
confirmed(['Make Example',output_is,'out1']).
confirmed(['Make Example',output_is,'out2']).
confirmed(['Make Example', has_output_number, 2]).
confirmed(['Make Example',control_is,'con1']).
confirmed(['Make Example',control_is,'con2']).
confirmed(['Make Example', has_control_number, 2]).
confirmed(['Make Example',mechanism_is,'Mechanism1']).
confirmed(['Make Example', has_mechanism_number, 1]).
```

Figure E.3. Predicate Data File Produced by *Save (.pro)* (parent)

```
confirmed([title, is, 'Make Example']).
confirmed([node, is, 'A1']).
confirmed([activityname, is, 'box1']).
confirmed(['box1',number_is,1]).
confirmed(['box1',input_is,'in2']).
confirmed(['box1',input_is,'In1']).
confirmed(['box1', has_input_number, 2]).
confirmed(['box1',output_is,'out2']).
confirmed(['box1', has_output_number, 1]).
confirmed(['box1',control_is,'con1']).
confirmed(['box1', has_control_number, 1]).
confirmed(['box1',mechanism_is,null]).
confirmed([activityname, is, 'box2']).
confirmed(['box2',number_is,2]).
confirmed(['box2',input_is,'out2']).
confirmed(['box2',input_is,'in3']).
confirmed(['box2', has_input_number, 2]).
confirmed(['box2',output_is,'in2']).
confirmed(['box2',output_is,'out1']).
confirmed(['box2',output_is,'out4']).
confirmed(['box2', has_output_number, 3]).
confirmed(['box2',control_is,'con2']).
confirmed(['box2',control_is,'con3']).
confirmed(['box2', has_control_number, 2]).
confirmed(['box2',mechanism_is,null]).
confirmed([activityname, is, 'box3']).
confirmed(['box3',number_is,3]).
confirmed(['box3',input_is,'in4']).
confirmed(['box3', has_input_number, 1]).
confirmed(['box3',output_is,'out2']).
confirmed(['box3',output_is,'con3']).
confirmed(['box3', has_output_number, 2]).
confirmed(['box3',control_is,'out4']).
confirmed(['box3', has_control_number, 1]).
confirmed(['box3',mechanism_is,'Mechanism1']).
confirmed(['box3', has_mechanism_number, 1]).
```

Figure E.4. Predicate Data File Produced by *Save (.pro)* (child)

```
confirmed([title, is, 'Make Example']).
confirmed([node, is, 'A1']).
confirmed([activityname, is, 'box1']).
confirmed(['box1',number_is,1]).
confirmed(['box1',input_is,'in2']).
confirmed(['box1',input_is,'In1']).
confirmed(['box1', has_input_number, 2]).
confirmed(['box1',output_is,'out2']).
confirmed(['box1', has_output_number, 1]).
confirmed(['box1',control_is,'con1']).
confirmed(['box1', has_control_number, 1]).
confirmed(['box1',mechanism_is,null]).
```

Figure E.5. Predicate Data File Produced by *Activity*

```
confirmed([title, is, 'Make Example']).
confirmed([node, is, 'AO']).
confirmed([activityname, is, 'Make Example']).
confirmed(['Make Example',number_is,0]).
confirmed(['Make Example',input_is,'In1']).
confirmed(['Make Example',input_is,'in2']).
confirmed(['Make Example', has_input_number, 2]).
confirmed(['Make Example',output_is,'out1']).
confirmed(['Make Example',output_is,'out2']).
confirmed(['Make Example', has_output_number, 2]).
confirmed(['Make Example',control_is,'con1']).
confirmed(['Make Example',control_is,'con2']).
confirmed(['Make Example', has_control_number, 2]).
confirmed(['Make Example',mechanism_is,'Mechanism1']).
confirmed(['Make Example', has_mechanism_number, 1]).
confirmed([child_title, is, 'Make Example']).
confirmed([boundary_control1, is, 'con1']).
confirmed([boundary_output1, is, 'out1']).
confirmed([boundary_output2, is, 'out2']).
confirmed([boundary_input1, is, 'in2']).
confirmed([boundary_control2, is, 'con2']).
confirmed([boundary_input2, is, 'In1']).
confirmed([boundary_mechanism1, is, 'Mechanism1']).
confirmed([boundary_input, has_number, 2]).
confirmed([boundary_output, has_number, 2]).
confirmed([boundary_control, has_number, 2]).
confirmed([boundary_mechanism, has_number, 1]).
```

Figure E.6. Predicate Data File Produced by *Boundary*

# Appendix F. *Source Code*

The purpose of this appendix represents the source code which was implemented during this thesis investigation. This appendix contains two main source codes: one for IDEF$_0$ Diagram Translator, the other for IDEF$_0$ Syntax Expert System.

## *Table of Contents*

### *IDEF$_0$ Diagram Translator*

The purpose of this section is to provide the source code documentation of the IDEF$_0$ Diagram Translator. The documentation conformed to the software engineering standards in AFIT's *System Development Documentation Guidelines and Standards* draft #4 (8).

```
/*************************************************************************
 *                                                                      *
 *   DATE:    3  Feb 1990                                               *
 *   VERSION: 1.0                                                       *
 *                                                                      *
 *   TITLE:   IDEF0 Diagram Translator                                  *
 *   FILENAME: syntaxfuncs.c                                            *
 *   DESCRIPTION:                                                       *
 *        This file provides a means of translating the IDEF0 diagram   *
 *        into a set of predicate forms and generating the predicates   *
 *        files as the data base of IDEF0 Syntax Expert System.         *
 *   PROJECT: AI                                                        *
 *   OPERATING SYSTEM:  UNIX 4.3                                        *
 *   LANGUAGE:          C                                               *
 *   FILE PROCESSING: Must compile with SAtool.                         *
 *   CONTENTS: check_input_abox(), single_headed_input(),               *
 *             double_headed_input(), double_headed_input_with_slash(), *
 *             check_output_abox(), single_headed_output(),             *
 *             double_headed_output_with_control(),                     *
 *             double_headed_output_with_input(), double_headed_output()*
 *             check_control_abox(), single_headed_control(),           *
 *             double_headed_control_with_slash(),                      *
 *             double_headed_control(), check_mechanism_abox(),         *
 *             single_headed_mechanism(),search_labels_touched_abox(),  *
 *             get_labels_for_abox(), save_arrow_info_of_abox(),        *
 *             create_temp_box_info(), find_clicked_box(),              *
 *             check_button_for_activity(), check_activity(),           *
 *             create_temp_boundary_info(), check_parent_box_file(),    *
 *             check_button_for_boundary(), check_boundary(),           *
 *             save_null_boundary(), search_NLR_boundary_line_info(),   *
 *             search_boundary_info(), save_boundary_line_info(),       *
 *             save_header_info(), traverse_boxes(), store_predicates(),*
 *             overwrite_predicates(), get_filename_for_predicates()    *
 *             save_predicates()                                        *
 *   AUTHOR:  Intaek Kim                                                *
 *   HISTORY:                                                           *
 *            10/Jan/90 : Modify the print out format                   *
 *            02/Feb/90 : Add save_header_info()                        *
 *            04/Feb/90 : Add save_boundary_info()                      *
 ************************************************************************/

#include <stdio.h>
#include <suntool/sunview.h>
#include <suntool/canvas.h>
#include <suntool/panel.h>
```

```
#include <suntool/textsw.h>
#include <sys/param.h>
#include "globals.h"

/********************** GLOBALS TO THIS FILE ****************************/
    int     number_of_boundary_input, number_of_boundary_output;
    int     number_of_boundary_control, number_of_boundary_mechanism;
    char    last_file_name[FILE_NAME_LENGTH +5] = "";
    struct text_line_struct *Line_labels;
```

```
/************************************************************************
 *                                                                      *
 *   DATE: 15 Feb 1990                                                  *
 *   VERSION: 1.0                                                       *
 *                                                                      *
 *   NAME: check_input_abox()                                           *
 *   MODULE NUMBER:                                                     *
 *   DESCRIPTION:                                                       *
 *              This module provides a means of generating a linked     *
 *              list, Line_labels, with all information labels of input *
 *              arrows attatched on an activity box.                    *
 *   ALGORITHM:                                                         *
 *   PASSED VARIABLES:    tem_line                                      *
 *                                                                      *
 *   RETURNS: None                                                      *
 *   GLOBAL VARIABLES USED:  Line_labels                                *
 *   GLOVAL VARIABLES CHANGED: Line_labels                              *
 *   FILES READ: None                                                   *
 *   FILES WRITTEN: None                                                *
 *   HARDWARE INPUT: None                                               *
 *   HARDWARE OUTPUT:None                                               *
 *   MODULES CALLED: signle_headed_input(), double_headed_input()       *
 *                   double_headed_input_with_slash()                   *
 *   CALLING MODULES: search_labels_touched_abox()                      *
 *                                                                      *
 *   AUTHOR: Intaek Kim                                                 *
 *   HISTORY:                                                           *
 *   ABSTRACT DATA TYPE :                                               *
 *   ORDER OF :                                                         *
 ***********************************************************************/
void
check_input_abox(tem_line)
struct line_struct *tem_line;
   {
extern struct text_line_struct *Line_labels;
extern add_to_inputs_tree();
char buf[DESCRIPTION_LINE_LENGTH+1];

    single_headed_input(tem_line);
    double_headed_input(tem_line);
    double_headed_input_with_slash(tem_line);
    return;
    }
```

```
/*************************************************************************
 * DATE: 25 Feb 1990                                                     *
 * VERSION: 1.0                                                          *
 *                                                                       *
 * NAME: single_headed_input()                                           *
 * MODULE NUMBER:                                                        *
 * DESCRIPTION: This module provides a means of hooking the line         *
 *              labels of input arrows with single head to a linked      *
 *              list, Line_label.                                        *
 * ALGROTHM:                                                             *
 * PASSED VARIABLES:  tem_line                                           *
 * RETURNS: None                                                         *
 * GLOBAL VARIABLES USED:  box,line,Line_labels                          *
 * GLOBAL VARIABLES CHANGED:  Line_labels                                *
 * FILES READ: None                                                      *
 * FILES WRITTEN: None                                                   *
 * HARDWARE INPUT: None                                                  *
 * HARDWARE OUTPUT: None                                                 *
 * MODULES CALLED: get_closest_label(), get_branchnode(),                *
 *                 find_TO_ALL_branchnode(), find_branchnode(),          *
 *                 add_to_inputs_tree()                                  *
 * CALLING MODULES: check_input_abcx()                                   *
 * AUTHOR: Intaek Kim                                                    *
 * HISTORY:                                                              *
 * ABSTRACT DATA TYPE:                                                   *
 * ORDER OF:                                                             *
 ************************************************************************/
 single_headed_input(tem_line)
 struct line_struct *tem_line;
   {
 extern get_closest_label(),get_branchnode(),find_TO_ALL_branchnode();
 extern find_branchnode(),add_to_inputs_tree();
 extern struct box_struct *box;
 extern struct line_struct *line;
 struct line_struct *original_line;
 char   buf[DESCRIPTION_LINE_LENGTH +1];
 int    original_line_type;

    if( /* check if  tem_line is touched on the left side of */
/* box with 10 deviation.                                  */
      (tem_line->end_position.x >= box->swcorner.x-10) &&
      (tem_line->end_position.x <= box->swcorner.x)    &&
      (tem_line->end_position.y <= box->swcorner.y)    &&
      (tem_line->end_position.y >= box->swcorner.y-BOX_HT) &&
      ( /* check if tem_line has an end arrow */
        (tem_line->end_activity_num & (ARROW_HEAD)) != 0) &&
       ((tem_line->end_activity_num & (DOT_B_L|DOT_T_R)) == 0)) {
    /* This tem_line is an input arrow for temp_box with some        */
    /* toralence, i.e. not require the line must be touched box     */
    /* so as to be an input arrow(single headed arrow).             */
      original_line_type = get_branchnode(tem_line);
      original_line = (struct line_struct *)
      find_branchnode(original_line_type);
      if(get_closest_label(original_line,buf,tem_line->struct_type)
  == MYTRUE)
{ /* tem_line has a line label */
 Line_labels = (struct text_line_struct *)
add_to_inputs_tree(Line_labels,buf);
 return;
 }
    /* No line label on tem_line - Check if tem_line is a FROM_ALL */
    /* type line. If so, search TO_ALL and its label                */
      if(original_line->start_activity_num == FROM_ALL) {
       original_line = (struct line_struct *)
       find_TO_ALL_branchnode(original_line->to_from_label);
       original_line_type = get_branchnode(original_line);
       original_line = (struct line_struct *)
       find_branchnode(original_line_type);
       if(get_closest_label(original_line,buf,line->struct_type)
  == MYTRUE)
 {
  Line_labels = (struct text_line_struct *)
         add_to_inputs_tree(Line_labels,buf);
  return;
  }
        }
     strcpy(buf,"");
     Line_labels = (struct text_line_struct *)
    add_to_inputs_tree(Line_labels,buf);
    }
   return;
  }
```

```
/***********************************************************************
 *                                                                     *
 * DATE: 16 Feb 1990                                                   *
 * VERSION: 1.0                                                        *
 * NAME : double_headed_input()                                        *
 * MODULE NUMBER:                                                      *
 * DESCRIPTION: This module provides a linked list(Line_labels) which *
 *              has the labels of the input arrows touched on an       *
 *              activity box with double head.                         *
 * ALGORITHM:                                                          *
 * PASSED VARIABLES: tem_line                                          *
 * RETURNS: None                                                       *
 * GLOBAL VARIABLES USED: box, line                                    *
 * GLOBAL VARIABLES CHANGED: Line_labels                               *
 * FILES READ: None                                                    *
 * FILES WRITTEN: None                                                 *
 * HARDWARE INPUT: None                                                *
 * HARDWARE OUTPUT: None                                               *
 * MODULES CALLED: get2_closest_label(), find_branchnode(),            *
 *                 find_TO_ALL_branchnode(), get_branchnode(),         *
 *                 add_to_inputs_tree()                                *
 * CALLING MODULES: check_input_abox()                                 *
 * AUTHOR: Intaek Kim                                                  *
 * HISTORY:                                                            *
 * ABSTRACT DATA TYPE:                                                 *
 * ORDER OF:                                                           *
 **********************************************************************/
 double_headed_input(tem_line)
 struct line_struct *tem_line;
   {
 extern get2_closest_label(),find_branchnode(),find_TO_ALL_branchnode();
 extern get_branchnode(),add_to_inputs_tree();
 extern struct box_struct *box;
 extern struct line_struct *line;
 struct line_struct *original_line;
 char   buf[DESCRIPTION_LINE_LENGTH +1];
 int    original_line_type;

    if((tem_line->end_position.x >= box->swcorner.x-10) &&
       (tem_line->end_position.x <= box->swcorner.x)    &&
       (tem_line->end_position.y <= box->swcorner.y)    &&
       (tem_line->end_position.y >= box->swcorner.y-BOX_HT) &&
       ( /* tem_line is an input arrow with a double headed arrow */
         /* in some extend(toralence).                             */
        (tem_line->end_activity_num & (ARROW_HEAD)) != 0)  &&
       ((tem_line->end_activity_num & (DOT_T_R)) != 0)) {
        original_line_type = get_branchnode(tem_line);
        original_line = (struct line_struct *)
 find_branchnode(original_line_type);
        if(/* Exists the line label on tem_line   */
   get2_closest_label(original_line,buf,tem_line->struct_type)
           == MYTRUE) {
           Line_labels = (struct text_line_struct *)
  add_to_inputs_tree(Line_labels,buf);
           return;
           }
      /* No line label on tem_line - Check if tem_line is a FROM_ALL */
      /* type line. If so, search TO_ALL and its label.             */
      if(original_line->start_activity_num == FROM_ALL) {
        original_line = (struct line_struct *)
find_TO_ALL_branchnode(original_line->to_from_label);
        original_line_type = get_branchnode(original_line);
        original_line = (struct line_struct *)
find_branchnode(original_line_type);
        if(get2_closest_label(original_line,buf,line->struct_type)
           == MYTRUE) {
           Line_labels = (struct text_line_struct *)
  add_to_inputs_tree(Line_labels,buf);
   return;
   }
          }
       strcpy(buf,"");
       Line_labels = (struct text_line_struct *)
      add_to_inputs_tree(Line_labels,buf);
      }
      return;
     }
```

```
/*********************************************************************
 *                                                                   *
 * DATE: 16 Feb 1990                                                 *
 * VERSION: 1.0                                                      *
 * NAME : double_headed_input_with_slash()                           *
 * MODULE NUMBER:                                                    *
 * DESCRIPTION: This module provides a means of adding the line      *
 *              labels with a slash of input arrows with double head *
 *              to Line_labels structure                             *
 * ALGORITHM:                                                        *
 * PASSED VARIABLES: tem_line                                        *
 * RETURNS: None                                                     *
 * GLOBAL VARIABLES USED: box                                        *
 * GLOBAL VARIABLES CHANGED: Line_labels                             *
 * FILES READ: None                                                  *
 * FILES WRITTEN: None                                               *
 * HARDWARE INPUT: None                                              *
 * HARDWARE OUTPUT: None                                             *
 * MODULES CALLED: get3_first_label(), add_to_inputs_tree()          *
 * CALLING MODULES: check_input_abox()                               *
 * AUTHOR: Intaek Kim                                                *
 * HISTORY:                                                          *
 * ABSTRACT DATA TYPE:                                               *
 * ORDER OF:                                                         *
 ********************************************************************/
double_headed_input_with_slash(tem_line)
struct line_struct *tem_line;
   {
extern get3_first_label(),add_to_inputs_tree();
extern struct box_struct *box;
char   buf[DESCRIPTION_LINE_LENGTH +1];

    if((tem_line->start_position.x < box->swcorner.x+BOX_WIDTH+15) &&
       (tem_line->start_position.x >= box->swcorner.x+BOX_WIDTH)  &&
       (tem_line->start_position.y <= box->swcorner.y)          &&
       ((tem_line->start_activity_num & (ARROW_HEAD)) !=0)      &&
       ((tem_line->start_activity_num & (DOT_B_L)) != 0))
       {
if(get3_first_label(tem_line,buf) == MYTRUE) {
  Line_labels = (struct text_line_struct *)
 add_to_inputs_tree(Line_labels,buf);
  return;
  }
        strcpy(buf,"");
Line_labels = (struct text_line_struct *)
```

```
       add_to_inputs_tree(Line_labels,buf);
}
     return;
    }
```

```
/*********************************************************************
 *                                                                   *
 * DATE: 17 Feb 1990                                                 *
 * VERSION: 1.0                                                      *
 * NAME: check_output_abox()                                         *
 * MODULE NUMBER:                                                    *
 * DESCRIPTION:                                                      *
 *             This module provides a means of generating a linked   *
 *             list, Line_labels, with all information labels of    *
 *             output arrows attatched on an activity box.           *
 * ALGORITHM:                                                        *
 * PASSED VARIABLES: tem_line                                        *
 * RETURNS: None                                                     *
 * GLOBAL VARIABLES USED: Line_labels                                *
 * GLOBAL VARIABLES CHANGED: Line_labels                             *
 * FILES READ: None                                                  *
 * FILES WRITTEN: None                                               *
 * HARDWARE INPUT: None                                              *
 * HARDWARE OUTPUT: None                                             *
 * MODULES CALLED: single_headed_output(), double_headed_output()    *
 *                 double_headed_output_with_input()                 *
 *                 double_headed_output_with_control()               *
 * CALLING MODULES: search_labels_touched_abox()                     *
 * AUTHOR: Intaek Kim                                                *
 * HISTORY:                                                          *
 * ABSTRACT DATA TYPE:                                               *
 * ORDER OF:                                                         *
 ********************************************************************/
 void
 check_output_abox(tem_line)
 struct line_struct *tem_line;
   {
    extern struct text_line_struct *Line_labels;

     single_headed_output(tem_line);
     double_headed_output_with_control(tem_line);
     double_headed_output_with_input(tem_line);
     double_headed_output(tem_line);
     return;
   }
```

```
/**********************************************************************
 *                                                                   *
 * DATE: 17 Feb 1990                                                 *
 * VERSION: 1.0                                                      *
 * NAME: single_headed_output()                                      *
 * MODULE NUMBER:                                                    *
 * DESCRIPTION:                                                      *
 *            This module provides a means of adding line labels of *
 *            output arrows with single head.                        *
 * ALGORITHM:                                                        *
 * PASSED VARIABLES: tem_line                                        *
 * RETURNS: None                                                     *
 * GLOBAL VARIABLES USED: Line_labels                                *
 * GLOBAL VARIABLES CHANGED: Line_labels                             *
 * FILES READ: None                                                  *
 * FILES WRITTEN: None                                               *
 * HARDWARE INPUT: None                                              *
 * HARDWARE OUTPUT: None                                             *
 * MODULES CALLED: get_first_label(), add_to_inputs_tree()           *
 * CALLING MODULES: check_output_abox()                              *
 * AUTHOR: Intaek Kim                                                *
 * HISTORY:                                                          *
 * ABSTRACT DATA TYPE:                                               *
 * ORDER OF:                                                         *
*********************************************************************/
 single_headed_output(tem_line)
 struct line_struct *tem_line;
   {
    extern get_first_label(),add_to_inputs_tree();
    char buf[DESCRIPTION_LINE_LENGTH +1];

    if(/* Output leaves the right side of a box - single headed arrow */
       (tem_line->start_position.x < box->swcorner.x+BOX_WIDTH+15) &&
       (tem_line->start_position.x >= box->swcorner.x+BOX_WIDTH)   &&
       (tem_line->start_position.y <= box->swcorner.y)             &&
       (tem_line->start_position.y >= box->swcorner.y-BOX_HT)      &&
       ((tem_line->start_activity_num & (ARROW_HEAD)) == 0))
       {
        if(get_first_label(tem_line,buf) == MYTRUE) {
          Line_labels = (struct text_line_struct *)
 add_to_inputs_tree(Line_labels,buf);
          return;
  }
        strcpy(buf,"");
Line_labels = (struct text_line_struct *)
```

```
  add_to_inputs_tree(Line_labels,buf);
  }
 return;
}
```

```
/*********************************************************************
 *                                                                   *
 * DATE: 17 Feb 1990                                                 *
 * VERSION: 1.0                                                      *
 * NAME: double_headed_output_with_control()                         *
 * MODULE NUMBER:                                                    *
 * DESCRIPTION:                                                      *
 *             This module provides a means of adding the line labels*
 *             of the output arrows with double heads which become   *
 *             controls of another activity box to a linked list.    *
 * ALGORITHM:                                                        *
 * PASSED VARIABLES: tem_line                                        *
 * RETURNS: None                                                     *
 * GLOBAL VARIABLES USED: line, Line_labels                          *
 * GLOBAL VARIABLES CHANGED: Line_labels                             *
 * FILES READ: None                                                  *
 * FILES WRITTEN: None                                               *
 * HARDWARE INPUT: None                                              *
 * HARDWARE OUTPUT: None                                             *
 * MODULES CALLED: get_branchnode(), find_branchnode(),              *
 *                 get3_closest_label(), find_TO_ALL_branchnode()    *
 *                 add_to_inputs_tree()                              *
 * CALLING MODULES: check_output_abox()                              *
 * AUTHOR: Intaek Kim                                                *
 * HISTORY:                                                          *
 * ABSTRACT DATA TYPE:                                               *
 * ORDER OF:                                                         *
*********************************************************************/
double_headed_output_with_control(tem_line)
struct line_struct *tem_line;
  {
   extern get_branchnode(),find_branchnode(),get3_closest_label();
   extern find_TO_ALL_branchnode();
   extern struct line_struct *line;
   struct line_struct *original_line;
   char buf[DESCRIPTION_LINE_LENGTH+1];
   int original_line_type;

   if(/* Output with a double headed control arrow - need to    */
      /* search the tem_line for the closest label with a slash */
      /* and get the label after the slash.                     */
      (tem_line->end_position.y > box->swcorner.y-BOX_HT-15)    &&
      (tem_line->end_position.y <= box->swcorner.y-BOX_HT)      &&
      (tem_line->end_position.x <= box->swcorner.x+BOX_WIDTH)   &&
      (tem_line->end_position.x >= box->swcorner.x)             &&
       ((tem_line->end_activity_num & (ARROW_HEAD)) != 0)        &&
       ((tem_line->end_activity_num & (DOT_B_L)) != 0))
       {
original_line_type = get_branchnode(tem_line);
        original_line = (struct line_struct *)
find_branchnode(original_line_type);
        if(get3_closest_label(original_line,buf,tem_line->struct_type)
  == MYTRUE) {
  Line_labels = (struct text_line_struct *)
 add_to_inputs_tree(Line_labels,buf);
  return;
  }
        /* No line label on tem_line - Check if tem_line is a       */
/* FROM_ALL type output. If so, search TO_ALL and its label.*/
if(original_line->start_activity_num == FROM_ALL) {
  original_line = (struct line_struct *)
  find_TO_ALL_branchnode(original_line->to_from_label);
          original_line_type = get_branchnode(original_line);
  original_line = (struct line_struct *)
  find_branchnode(original_line_type);
  if(/*have found the branchnode of the TO_ALL line segment*/
     get3_closest_label(original_line,buf,line->struct_type))
             {
      Line_labels = (struct text_line_struct *)
     add_to_inputs_tree(Line_labels,buf);
      return;
      }
          }
         strcpy(buf,"");
 Line_labels = (struct text_line_struct *)
add_to_inputs_tree(Line_labels,buf);
        }
       return;
      }
```

```
/********************************************************************
 * DATE: 17 Feb 1990                                                *
 * VERSION: 1.0                                                     *
 * NAME : double_headed_output_with_input()                         *
 * MODULE NUMBER:                                                   *
 * DESCRIPTION: This module provides a means of adding the line     *
 *              labels of the output with double heads and the slash*
 *              to a linked list, Line_labels.                      *
 * ALGORITHM:                                                       *
 * PASSED VARIABLES: tem_line                                       *
 * RETURNS: None                                                    *
 * GLOBAL VARIABLES USED: line                                      *
 * GLOBAL VARIABLES CHANGED: Line_labels                            *
 * FILE READ: None                                                  *
 * FILES WRITTEN: None                                              *
 * HARDWARE INPUT: None                                             *
 * HARDWARE OUTPUT: None                                            *
 * MODULES CALLED: get_branchnode(), find_branchnode(),             *
 *                 get3_closest_label(), add_to_inputs_tree()       *
 *                 find_TO_ALL_branchnode()                         *
 * CALLING MODULES: check_output_abox()                             *
 * AUTHOR: Intaek Kim                                               *
 * HISTORY:                                                         *
 * ABSTRACT DATA TYPE:                                              *
 * ORDER OF:                                                        *
 *******************************************************************/
 double_headed_output_with_input(tem_line)
 struct line_struct *tem_line;
  {
   extern get_branchnode(),find_branchnode(),get3_closest_label();
   extern add_to_inputs_tree(),find_TO_ALL_branchnode();
   extern struct line_struct *line;
   struct line_struct *original_line;
   int original_line_type;
   char buf[DESCRIPTION_LINE_LENGTH +1];

   if(/* tem_line is an output with a double headed input */
      /* arrow on a box */
      (tem_line->end_position.x >= box->swcorner.x -15)    &&
      (tem_line->end_position.x <= box->swcorner.x)        &&
      (tem_line->end_position.y <= box->swcorner.y)        &&
      (tem_line->end_position.y >= box->swcorner.y -BOX_HT)&&
      ((tem_line->end_activity_num & (ARROW_HEAD)) != 0)   &&
      ((tem_line->end_activity_num & (DOT_T_R))    != 0))
      {
       original_line_type = get_branchnode(tem_line);
       original_line = (struct line_struct *)
                find_branchnode(original_line_type);
       if(/* Exists label on the original line */
  get3_closest_label(original_line,buf,tem_line->struct_type)
  == MYTRUE) {
  Line_labels = (struct text_line_struct *)
 add_to_inputs_tree(Line_labels,buf);
  return;
  }
       if(/* No line label on tem_line - Check if      */
  /* tem_line is a type of FROM_ALL            */
  /* line. If so, Search TO_ALL and its label. */
  original_line->start_activity_num == FROM_ALL)
  {
   original_line = (struct line_struct *)
find_TO_ALL_branchnode(original_line->to_from_label);
           original_line_type = get_branchnode(original_line);
   original_line = (struct line_struct *)
            find_branchnode(original_line_type);
   if(/* Find the branchnode of the TO_ALL line segment */
      get3_closest_label(original_line,buf,line->struct_type)
      == MYTRUE)
      {
               Line_labels = (struct text_line_struct *)
      add_to_inputs_tree(Line_labels,buf);
               return;
       }
           }
          strcpy(buf,"");
  Line_labels = (struct text_line_struct *)
 add_to_inputs_tree(Line_labels,buf);
         }
        return;
       }
```

```
/*********************************************************************
 *                                                                   *
 * DATE: 17 Feb 1990                                                 *
 * VERSION: 1.0                                                      *
 * NAME: double_headed_output()                                      *
 * MODULE NUMBER:                                                    *
 * DESCRIPTION: This module provides a means of adding the line      *
 *              labels of the output with double heads to a linked   *
 *              list, Line_labels.                                   *
 * ALGORITHM:                                                        *
 * PASSED VARIABLES: tem_line                                        *
 * RETURNS: None                                                     *
 * GLOBAL VARIABLES USED: Line_labels                                *
 * GLOBAL VARIABLES CHANGED: Line_labels                             *
 * FILES READ: None                                                  *
 * FILES WRITTEN: None                                               *
 * HARDWARE INPUT: None                                              *
 * HAREWARE OUTPUT: None                                             *
 * MODULES CALLED: get2_first_label(), add_to_inputs_tree()          *
 * CALLING MODULES: check_output_abox()                              *
 * AUTHOR: Intaek Kim                                                *
 * HISTORY:                                                          *
 * ABSTRACT DATA TYPE:                                               *
 * ORDER OF:                                                         *
 ********************************************************************/
 double_headed_output(tem_line)
 struct line_struct *tem_line;
  {
   extern get2_first_label(),add_to_inputs_tree();
   char buf[DESCRIPTION_LINE_LENGTH +1];

   if(/* Output line leaves the right side of a box and there is */
      /* a double headed arrow.                                   */
      (tem_line->start_position.x <= box->swcorner.x + BOX_WIDTH + 10) &&
      (tem_line->start_position.x >= box->swcorner.x + BOX_WIDTH)      &&
      (tem_line->start_position.y <= box->swcorner.y)                  &&
      (tem_line->start_position.y >= box->swcorner.y - BOX_HT)         &&
      ((tem_line->start_activity_num & (ARROW_HEAD)) != 0)             &&
      ((tem_line->start_activity_num & (DOT_B_L|DOT_T_R)) != 0))
      {
       if(get2_first_label(tem_line,buf) == MYTRUE) {
          Line_labels = (struct text_line_struct *)
                        add_to_inputs_tree(Line_labels,buf);
          return;
 }
```

```
  strcpy(buf,"");
  Line_labels = (struct text_line_struct *)
               add_to_inputs_tree(Line_labels,buf);
 }
 return;
}
```

```
/*********************************************************************
 *                                                                   *
 * DATE: 19 Feb 1990                                                 *
 * VERSION: 1.0                                                      *
 * NAME: check_control_abox()                                        *
 * MODULE NUMBER:                                                    *
 * DESCRIPTION: This module provides a means of generating a linked *
 *              list(Line_labels) which has the information of labels*
 *              of the control arrows attatched on an activity box.  *
 * ALGORITHM:                                                        *
 * PASSED VARIABLES: tem_line                                        *
 * RETURNS: None                                                     *
 * GLOBAL VARIABLES USED: None                                       *
 * GLOBAL VARIABLES CHANGED: None                                    *
 * FILES READ: None                                                  *
 * FILES WRITTEN: None                                               *
 * HARDWARE INPUT: None                                              *
 * HARDWARE OUTPUT: None                                             *
 * MODULES CALLED: single_headed_control(), double_headed_control() *
 *                 double_headed_control_with_slash()                *
 * CALLING MODULES: search_labels_touched_abox()                     *
 * AUTHOR: Intaek Kim                                                *
 * HISTORY:                                                          *
 * ABSTRACT DATA TYPE:                                               *
 * ORDER OF:                                                         *
 *********************************************************************/
void
check_control_abox(tem_line)
struct line_struct *tem_line;
 {
  extern struct text_line_struct *Line_labels;

   single_headed_control(tem_line);
   double_headed_control(tem_line);
   double_headed_control_with_slash(tem_line);
   return;
  }
```

```
/*********************************************************************
 *                                                                   *
 * DATE: 19 Feb 1990                                                 *
 * VERSION: 1.0                                                      *
 * NAME: single_headed_control()                                     *
 * MODULE NUMBER:                                                    *
 * DESCRIPTION: This module provides a means of adding the line      *
 *             labels of the control arrows with a single head to    *
 *             the linked list, Line_labels.                         *
 * ALGORITHM:                                                        *
 * PASSED VARIABLES: tem_line                                        *
 * RETURNS: None                                                     *
 * GLOBAL VARIABLES USED: line, Line_labels                          *
 * GLOBAL VARIABLES CHANGED: Line_labels                             *
 * FILES READ: None                                                  *
 * FILES WRITTEN: None                                               *
 * HARDWARE INPUT: None                                              *
 * HARDWARE OUTPUT: None                                             *
 * MODULES CALLED: get_closest_label(), add_to_inputs_tree()         *
 *                 get_branchnode(), find_branchnode(),              *
 *                 find_TO_ALL_branchnode()                          *
 * CALLING MODULES: check_control_abox()                             *
 * AUTHOR: Intaek Kim                                                *
 * HISTORY:                                                          *
 * ABSTRACT DATA TYPE:                                               *
 * ORDER OF:                                                         *
 ********************************************************************/
 single_headed_control(tem_line)
 struct line_struct *tem_line;
  {
   extern get_closest_label(),add_to_inputs_tree(),get_branchnode();
   extern find_branchnode(),find_TO_ALL_branchnode();
   extern struct line_struct *line;
   struct line_struct *original_line;
   int original_line_type;
   char buf[DESCRIPTION_LINE_LENGTH +1];

   if(/* Control line comes to the upper side of a box and there  */
      /* is a single headed arrow.                                */
      (tem_line->end_position.x <= box->swcorner.x + BOX_WIDTH) &&
      (tem_line->end_position.x >= box->swcorner.x)             &&
      (tem_line->end_position.y >= box->swcorner.y - BOX_HT - 10) &&
      (tem_line->end_position.y <= box->swcorner.y - BOX_HT)    &&
      ((tem_line->end_activity_num & (ARROW_HEAD)) != 0)        &&
      ((tem_line->end_activity_num & (DOT_B_L|DOT_T_R)) == 0))
```

```
    {
     original_line_type = get_branchnode(tem_line);
     original_line = (struct line_struct *)
                      find_branchnode(original_line_type);
     if(/* found a label on the line */
        get_closest_label(original_line,buf,tem_line->struct_type)
        == MYTRUE) {
         Line_labels = (struct text_line_struct *)
                        add_to_inputs_tree(Line_labels,buf);
return;
}
     if(/* No label on tem_line - Check if a FROM_ALL exists.  */
/* If so, search TO_ALL and get a label for this line. */
original_line->start_activity_num == FROM_ALL) {
original_line = (struct line_struct *)
              find_TO_ALL_branchnode(original_line->to_from_label);
        original_line_type = get_branchnode(original_line);
original_line = (struct line_struct *)
                         find_branchnode(original_line_type);
        if(/* found the branchnode of the TO_ALL line segment */
   get_closest_label(original_line,buf,line->struct_type)
   == MYTRUE) {
   Line_labels = (struct text_line_struct *)
                          add_to_inputs_tree(Line_labels,buf);
           return;
   }
         }
        strcpy(buf,"");
Line_labels = (struct text_line_struct *)
                       add_to_inputs_tree(Line_labels,buf);
       }
      return;
     }
```

```
/*********************************************************************
 *                                                                   *
 * DATE: 19 Feb 1990                                                 *
 * VERSION: 1.0                                                      *
 * NAME : double_headed_control_with_slash()                         *
 * MODULE NUMBER:                                                    *
 * DESCRIPTION: This module provides a means of adding the labels    *
 *             of the control arrows with double heads and the       *
 *             slash to the linked list, Line_labels.                *
 * ALGORITHM:                                                        *
 * PASSED VARIABLES: tem_line                                        *
 * RETURNS: None                                                     *
 * GLOBAL VARIABLES USED: Line_labels                                *
 * GLOBAL VARIABLES CHANGED: Line_labels                             *
 * FILES READ: None                                                  *
 * FILES WRITTEN: None                                               *
 * HARDWARE INPUT: None                                              *
 * HARDWARE OUTPUT: None                                             *
 * MODULES CALLED: get2_closest_label(), add_to_inputs_tree(),       *
 *                 get_branchnode(), find_TO_ALL_branchnode(),       *
 *                 find_branchnode()                                 *
 * CALLING MODULES: check_control_abox()                             *
 * AUTHOR: Intaek Kim                                                *
 * HISTORY:                                                          *
 * ABSTRACT DATA TYPE:                                               *
 * ORDER OF:                                                         *
 *********************************************************************/
 double_headed_control_with_slash(tem_line)
 struct line_struct *tem_line;
  {
   extern get2_closest_label(),add_to_inputs_tree(),get_branchnode();
   extern find_TO_ALL_branchnode(),find_branchnode();
   struct line_struct *original_line;
   char buf[DESCRIPTION_LINE_LENGTH +1];
   int original_line_type;

   if(/* Have a control with a double headed arrow */
      (tem_line->end_position.y <= box->swcorner.y-BOX_HT)  &&
      (tem_line->end_position.y >= box->swcorner.y-BOX_HT-15) &&
      (tem_line->end_position.x <= box->swcorner.x+BOX_WIDTH) &&
      (tem_line->end_position.x >= box->swcorner.x) &&
      ((tem_line->end_activity_num & (ARROW_HEAD)) != 0) &&
      ((tem_line->end_activity_num & (DOT_B_L)) != 0))
      {
       original_line_type = get_branchnode(tem_line);
       original_line = (struct line_struct *)
                        find_branchnode(original_line_type);
       if(get2_closest_label(original_line,buf,tem_line->struct_type)
 == MYTRUE) {
 Line_labels = (struct text_line_struct *)
                        add_to_inputs_tree(Line_labels,buf);
         return;
 }
       if(/* No label on tem_line - Check if tem_line is a FROM_ALL */
  /* type line. If so, search the TO_ALL and get a label    */
  /* from TO_ALL line.                                      */
  original_line->start_activity_num == FROM_ALL) {
  original_line = (struct line_struct *)
                 find_TO_ALL_branchnode(original_line->to_from_label);
          original_line_type = get_branchnode(original_line);
  original_line = (struct line_struct *)
                           find_branchnode(original_line_type);
  if(get2_closest_label(original_line,buf,line->struct_type)
     == MYTRUE) {
     Line_labels = (struct text_line_struct *)
                            add_to_inputs_tree(Line_labels,buf);
             return;
     }
           }
        strcpy(buf,"");
Line_labels = (struct text_line_struct *)
                       add_to_inputs_tree(Line_labels,buf);
      }
      return;
     }
```

```
/********************************************************************
 *                                                                  *
 * DATE: 20 Feb 1990                                                *
 * VERSION: 1.0                                                     *
 * NAME: double_headed_control()                                    *
 * MODULE NUMBER:                                                   *
 * DESCRIPTION: This module provides a means of adding the line     *
 *              labels of the control arrows with double head to    *
 *              the linked list, Line_labels.                       *
 * ALGORITHM:                                                       *
 * PASSED VARIABLES: tem_line                                       *
 * RETURNS: None                                                    *
 * GLOBAL VARIABLES USED: Line_labels                               *
 * GLOBAL VARIABLES CHANGED: Line_labels                            *
 * FILES READ: None                                                 *
 * FILES WRITTEN: Nnone                                             *
 * HARDWARE INPUT: None                                             *
 * HARDWARE OUTPUT: None                                            *
 * MODULES CALLED: get3_first_label(), add_to_inputs_tree()         *
 * CALLING MODULES: check_control_abox()                            *
 * AUTHOR: Intaek Kim                                               *
 * HISTORY:                                                         *
 * ABSTRACT DATA TYPE:                                              *
 * ORDER OF:                                                        *
 *******************************************************************/
 double_headed_control(tem_line)
 struct line_struct *tem_line;
   {
    extern get3_first_label(),add_to_inputs_tree();
    char buf[DESCRIPTION_LINE_LENGTH +1];

    if((tem_line->start_position.x <= box->swcorner.x+BOX_WIDTH+10) &&
       (tem_line->start_position.x >= box->swcorner.x+BOX_WIDTH)    &&
       (tem_line->start_position.y <= box->swcorner.y)              &&
       (tem_line->start_position.y >= box->swcorner.y-BOX_HT)       &&
       ((tem_line->start_activity_num & (ARROW_HEAD)) != 0)         &&
       ((tem_line->start_activity_num & (DOT_T_R)) != 0))
       {
if(get3_first_label(tem_line,buf) == MYTRUE) {
   Line_labels = (struct text_line_struct *)
                          add_to_inputs_tree(Line_labels,buf);
           return;
   }
         strcpy(buf,"");
         Line_labels = (struct text_line_struct *)
```

```
                    add_to_inputs_tree(Line_labels,buf);
  }
  return;
}
```

```
/*********************************************************************
 *                                                                   *
 * DATE: 21 Feb 1990                                                 *
 * VERSION: 1.0                                                      *
 * NAME: check_mechanism_abox()                                      *
 * MODULE NUMBER:                                                    *
 * DESCRIPTION: This module provides  means of generating a linked   *
 *             list, Line_labels, with all information labels of the *
 *             mechanism arrows attatched on an activity box.        *
 * ALGORITHM:                                                        *
 * PASSED VARIABLES: tem_line                                        *
 * RETURNS: None                                                     *
 * GLOBAL VARIABLES USED: Line_labels                                *
 * GLOBAL VARIABLES CHANGED: None                                    *
 * FILES READ: None                                                  *
 * FILES WRITTEN: None                                               *
 * HARDWARE INPUT: None                                              *
 * HARDWARE OUTPUT: None                                             *
 * MODULES CALLED: single_headed_mechanism()                         *
 * CALLING MODULES: search_labels_touched_abox()                     *
 * AUTHOR: Intaek Kim                                                *
 * HISTORY:                                                          *
 * ABSTRACT DATA TYPE:                                               *
 * ORDER OF:                                                         *
 ********************************************************************/
 void
 check_mechanism_abox(tem_line)
 struct line_struct *tem_line;
  {
   extern struct text_line_struct *Line_labels;

   single_headed_mechanism(tem_line);
   return;
  }
```

```
/**********************************************************************
 *                                                                    *
 * DATE: 21 Feb 1990                                                  *
 * VERSION: 1.0                                                       *
 * NAME: single_headed_mechanism()                                    *
 * MODULE NUMBER:                                                     *
 * DESCRIPTION: This module provides a means of adding the line       *
 *              labels of the mechanism arrows with a single head to  *
 *              a linked list, Line_labels.                           *
 * ALGORITHM:                                                         *
 * PASSED VARIABLES: tem_line                                         *
 * RETURNS: None                                                      *
 * GLOBAL VARIABLES USED: line, Line_labels                           *
 * GLOBAL VARIABLES CHANGED: Line_labels                              *
 * FILES READ: None                                                   *
 * FILES WRITTEN: None                                                *
 * HARDWARE INPUT: None                                               *
 * HARDWARE OUTPUT: None                                              *
 * MODULES CALLED: get_branchnode(), find_branchnode(),               *
 *                 get_closest_label(), find_TO_ALL_branchnode()      *
 *                 add_to_inputs_tree()                               *
 * CALLING MODULES: check_mechanism_abox()                            *
 * AUTHOR: Intaek Kim                                                 *
 * HISTORY:                                                           *
 * ABSTRACT DATA TYPE:                                                *
 * ORDER OF:                                                          *
 **********************************************************************/
 single_headed_mechanism(tem_line)
 struct line_struct *tem_line;
   {
    extern get_branchnode(),find_branchnode(),get_closest_label();
    extern find_TO_ALL_branchnode(),add_to_inputs_tree();
    extern struct line_struct *line;
    struct line_struct *original_line;
    int original_line_type;
    char buf[DESCRIPTION_LINE_LENGTH +1];

    if((tem_line->end_position.x <= box->swcorner.x+BOX_WIDTH)  &&
       (tem_line->end_position.x >= box->swcorner.x)            &&
       (tem_line->end_position.y >= box->swcorner.y)            &&
       (tem_line->end_position.y <= box->swcorner.y+15))
       {
        original_line_type = get_branchnode(tem_line);
        original_line = (struct line_struct *)
                          find_branchnode(original_line_type);
```

```
  if(get_closest_label(original_line,buf,tem_line->struct_type)
     == MYTRUE) {
     Line_labels = (struct text_line_struct *)
                    add_to_inputs_tree(Line_labels,buf);
     return;
     }
   if(original_line->start_activity_num == FROM_ALL) {
      original_line = (struct line_struct *)
          find_TO_ALL_branchnode(original_line->to_from_label);
     original_line_type = get_branchnode(original_line);
     original_line = (struct line_struct *)
                      find_branchnode(original_line_type);
     if(get_closest_label(original_line,buf,line->struct_type)
        == MYTRUE) {
        Line_labels = (struct text_line_struct *)
                       add_to_inputs_tree(Line_labels,buf);
        return;
        }
      }
   strcpy(buf,"");
   Line_labels = (struct text_line_struct *)
                  add_to_inputs_tree(Line_labels,buf);
   }
 return;
}
```

```
/***********************************************************************
 *                                                                     *
 *   DATE: 15 Feb 1990                                                 *
 *   VERSION: 1.0                                                      *
 *                                                                     *
 *   NAME: search_labels_touched_abox()                                *
 *   MODULE NUMBER:                                                    *
 *   DESCRIPTION:                                                      *
 *              This module provides a means of searching the line     *
 *              label in accordance with the ICOM code.                *
 *   ALGORITHM:                                                        *
 *   PASSED VARIABLES:   tem_line                                      *
 *                                                                     *
 *   RETURNS: None                                                     *
 *   GLOBAL VARIABLES USED:  Line_labels                               *
 *   GLOVAL VARIABLES CHANGED: None                                    *
 *   FILES READ: None                                                  *
 *   FILES WRITTEN:  None                                              *
 *   HARDWARE INPUT: None                                              *
 *   HARDWARE OUTPUT:None                                              *
 *   MODULES CALLED: check_input_abox(), check_output_abox(),          *
 *                   check_control_abox(),                             *
 *                   check_mechanism_abox(),                           *
 *   CALLING MODULES: get_labels_for_abox()                            *
 *                                                                     *
 *   AUTHOR: Intaek Kim                                                *
 *   HISTORY:                                                          *
 *   ABSTRACT DATA TYPE :                                              *
 *   ORDER OF :                                                        *
 **********************************************************************/
  search_labels_touched_abox(tem_line,indicator)
   struct line_struct *tem_line;
   char indicator;
    {
     extern struct text_line_struct *Line_labels;

     if(tem_line == NULL) return;
     else {
      search_labels_touched_abox(tem_line->left,indicator);
      search_labels_touched_abox(tem_line->right,indicator);

      switch(indicator) {
       case 'I':
        check_input_abox(tem_line);
break;
```

```
       case ’O’:
check_output_abox(tem_line);
break;
       case ’C’:
check_control_abox(tem_line);
break;
       case ’M’:
check_mechanism_abox(tem_line);
break;
       default:
break;
        }
      }
     return;
     }
```

```
/**********************************************************************
 *                                                                    *
 *   DATE: 15 Feb 1990                                                *
 *   VERSION: 1.0                                                     *
 *                                                                    *
 *   NAME: get_labels_for_abox                                        *
 *   MODULE NUMBER:                                                   *
 *   DESCRIPTION:                                                     *
 *         This module provides a means of constructing a linked list *
 *         (Line_labels) made of the line labels in accordance with   *
 *         the variable, indicator.                                   *
 *   ALGORITHM:                                                       *
 *   PASSED VARIABLES: indicator                                      *
 *                                                                    *
 *   RETURNS: None                                                    *
 *   GLOBAL VARIABLES USED:  line_rootnode, Line_labels               *
 *   GLOVAL VARIABLES CHANGED: Line_labels                            *
 *   FILES READ: None                                                 *
 *   FILES WRITTEN: None                                              *
 *   HARDWARE INPUT: none                                             *
 *   HARDWARE OUTPUT: None                                            *
 *   MODULES CALLED: search_labels_touched_abox()                     *
 *   CALLING MODULES: save_arrow_info_of_abox()                       *
 *                                                                    *
 *   AUTHOR: Intaek Kim                                               *
 *   HISTORY:                                                         *
 *   ABSTRACT DATA TYPE :                                             *
 *   ORDER OF :                                                       *
 **********************************************************************/
struct text_line_struct *
get_labels_for_abox(indicator)
 char indicator;
  {
   extern struct line_struct *line_rootnode;
   extern struct text_line_struct *Line_labels;
   struct line_struct *temporary_line;

   Line_labels = NULL;
   temporary_line = line_rootnode;
   while(temporary_line != NULL) {
     search_labels_touched_abox(temporary_line,indicator);
     temporary_line = temporary_line->next;
     }
   return(Line_labels);
  }
```

```
/**********************************************************************
 *                                                                    *
 *   DATE: 15 Feb 1990                                                *
 *   VERSION: 1.0                                                     *
 *                                                                    *
 *   NAME: save_arrow_info_of_abox()                                  *
 *   MODULE NUMBER:                                                   *
 *   DESCRIPTION:                                                     *
 *         This module provides a means of saving all information of  *
 *         arrows(ICOM) attatched on a box to a file.                 *
 *   ALGORITHM:                                                       *
 *   PASSED VARIABLES: fp, tem_box                                    *
 *                                                                    *
 *   RETURNS: None                                                    *
 *   GLOBAL VARIABLES USED: box                                       *
 *   GLOVAL VARIABLES CHANGED:   box                                  *
 *   FILES READ: None                                                 *
 *   FILES WRITTEN: *.pro, CHECKBOX.PRO, or CHECKBOUNDARY.PRO         *
 *   HARDWARE INPUT: None                                             *
 *   HARDWARE OUTPUT: None                                            *
 *   MODULES CALLED: get_labels_for_abox(), fprintf(), itoa()         *
 *   CALLING MODULES: create_temp_box_info(), traverse_boxes()        *
 *                                                                    *
 *   AUTHOR: Intaek Kim                                               *
 *   HISTORY:                                                         *
 *   ABSTRACT DATA TYPE :                                             *
 *   ORDER OF :                                                       *
 *********************************************************************/

 void
 save_arrow_info_of_abox(fp,tem_box)
  FILE    *fp;
  struct box_struct *tem_box;
   {
    extern itoa();
    extern struct box_struct *box;
    struct text_line_struct *ICOM_labels;
    char buf[DESCRIPTION_LINE_LENGTH+1];
    int  number_of_input = 0;
    int  number_of_output = 0;
    int  number_of_control = 0;
    int  number_of_mechanism = 0;

 /********** NAME ***********/
    fprintf(fp,"confirmed(['%s',is,in_data]).\n",
tem_box->name.text_string);

 /************* NUMBER ***********/
    itoa(tem_box->number,buf);
    fprintf(fp,"confirmed(['%s',number_is,%s]).\n",
    tem_box->name.text_string,buf);
    box = tem_box;

 /***************** INPUTS ****************/
    ICOM_labels = (struct text_line_struct *)get_labels_for_abox('I');
    if (ICOM_labels == NULL) {
      fprintf(fp,"confirmed(['%s',input_is,null]).\n",
      tem_box->name.text_string);
      }
    else {
     while(ICOM_labels != NULL) {
      fprintf(fp,"confirmed(['%s',input_is,'%s']).\n",
              tem_box->name.text_string, ICOM_labels->text_line);
      ICOM_labels = ICOM_labels->next;
      number_of_input += 1;
      }
     itoa(number_of_input,buf);
     fprintf(fp,"confirmed(['%s', has_input_number, %s]).\n",
             tem_box->name.text_string,buf);
     }

 /******************** OUTPUTS ************************/
    ICOM_labels = (struct text_line_struct *)get_labels_for_abox('O');
    if (ICOM_labels == NULL) {
      fprintf(fp,"confirmed(['%s',output_is,null]).\n",
              tem_box->name.text_string);
      }
    else {
     while(ICOM_labels != NULL) {
      fprintf(fp,"confirmed(['%s',output_is,'%s']).\n",
              tem_box->name.text_string,ICOM_labels->text_line);
      ICOM_labels = ICOM_labels->next;
      number_of_output += 1;
      }
     itoa(number_of_output,buf);
     fprintf(fp,"confirmed(['%s', has_output_number, %s]).\n",
             tem_box->name.text_string, buf);
    }
```

```
/********************** CONTROLS **********************/
   ICOM_labels = (struct text_line_struct *)get_labels_for_abox('C');
   if (ICOM_labels == NULL) {
     fprintf(fp,"confirmed(['%s',control_is,null]).\n",
             tem_box->name.text_string);
     }
   else {
    while(ICOM_labels != NULL) {
     fprintf(fp,"confirmed(['%s',control_is,'%s']).\n",
             tem_box->name.text_string,ICOM_labels->text_line);
     ICOM_labels = ICOM_labels->next;
     number_of_control += 1;
     }
    itoa(number_of_control,buf);
    fprintf(fp,"confirmed(['%s', has_control_number, %s]).\n",
            tem_box->name.text_string,buf);
    }


/****************** MECHANISMS ************************/
   ICOM_labels = (struct text_line_struct *)get_labels_for_abox('M');
   if (ICOM_labels == NULL) {
    fprintf(fp,"confirmed(['%s',mechanism_is,null]).\n",
            tem_box->name.text_string);
    }
   else {
    while(ICOM_labels != NULL) {
     fprintf(fp,"confirmed(['%s',mechanism_is,'%s']).\n",
             tem_box->name.text_string,ICOM_labels->text_line);
     ICOM_labels = ICOM_labels->next;
     number_of_mechanism += 1;
     }
    itoa(number_of_mechanism,buf);
    fprintf(fp,"confirmed(['%s', has_mechanism_number, %s]).\n",
            tem_box->name.text_string,buf);
    }
   return;
   }
```

```
/**********************************************************************
 *                                                                    *
 *   DATE: 15 Feb 1990                                                *
 *   VERSION: 1.0                                                     *
 *                                                                    *
 *   NAME: create_temp_box_info()                                     *
 *   MODULE NUMBER:                                                   *
 *   DESCRIPTION:                                                     *
 *                 This module provides a means of creating the       *
 *           temporary file, CHECKBOX.PRO, which contains a set of    *
 *           predicate forms for an activity box.                     *
 *   ALGORITHM:                                                       *
 *   PASSED VARIABLES: found_box                                      *
 *                                                                    *
 *   RETURNS: None                                                    *
 *   GLOBAL VARIABLES USED: None                                      *
 *   GLOVAL VARIABLES CHANGED: None                                   *
 *   FILES READ: None                                                 *
 *   FILES WRITTEN:  CHECKBOX.PRO                                     *
 *   HARDWARE INPUT: None                                             *
 *   HARDWARE OUTPUT: None                                            *
 *   MODULES CALLED: fopen(), put_message(), disable_input_window(),  *
 *                   save_header_info(), save_arrow_info_of_abox(),   *
 *                   fclose(), printf()                               *
 *   CALLING MODULES: find_clicked_box()                              *
 *                                                                    *
 *   AUTHOR: Intaek Kim                                               *
 *   HISTORY:                                                         *
 *   ABSTRACT DATA TYPE :                                             *
 *   ORDER OF :                                                       *
 **********************************************************************/
void
create_temp_box_info(found_box)
 struct box_struct *found_box;
 {
  extern put_message(),disable_input_window();
  FILE  *fp;

  if((fp = fopen("CHECKBOX.PRO","w")) == NULL) {
    put_message(1,"Unable to open the CHECKBOX.PRO file -- ABORT.");
    disable_input_window();
    }
  else {
    disable_input_window();
    save_header_info(fp);
```

```
   save_arrow_info_of_abox(fp,found_box);
   if(fclose(fp) != 0) printf("FILE CLOSE FAILED\n");
   }
 return;
}
```

```
/**********************************************************************
*                                                                    *
*   DATE: 15 Feb 1990                                                *
*   VERSION: 1.0                                                     *
*                                                                    *
*   NAME: find_clicked_box()                                         *
*   MODULE NUMBER:                                                   *
*   DESCRIPTION:                                                     *
*         This module provides a means of checking if there is a box *
*          within the cordinate(x,y) clicked by the user mouse.      *
*   ALGORITHM:                                                       *
*   PASSED VARIABLES: x,y                                            *
*                                                                    *
*   RETURNS: None                                                    *
*   GLOBAL VARIABLES USED: box_rootnode                              *
*   GLOVAL VARIABLES CHANGED: None                                   *
*   FILES READ: None                                                 *
*   FILES WRITTEN:  None                                             *
*   HARDWARE INPUT: None                                             *
*   HARDWARE OUTPUT: None                                            *
*   MODULES CALLED: put_message(), create_temp_box_info(),           *
*                   my_window_set(), null_proc()                     *
*   CALLING MODULES: check_button_for_activity()                     *
*                                                                    *
*   AUTHOR: Intaek Kim                                               *
*   HISTORY:                                                         *
*   ABSTRACT DATA TYPE :                                             *
*   CRDER OF :                                                       *
**********************************************************************/
void
find_clicked_box(x,y)
 int  x,y;
 {
  extern put_message(),my_window_set(),null_proc();
  extern struct box_struct *box_rootnode;
  struct box_struct *bbox;

  bbox = box_rootnode;
  while (bbox != NULL) {
   if(x >= bbox->swcorner.x && x <= bbox >swcorner.x + BOX_WIDTH &&
      y >= bbox->swcorner.y - BOX_HT && y <= bbox->swcorner.y)
      {
       put_message(1,"Enter the prolog environment
      using another window.");
       create_temp_box_info(bbox);
      my_window_set(null_proc);
      return;
     }
  bbox = bbox->next;
 }
 put_message(1,"Box not found - Try again(|L| to select)");
 return;
}
```

```
/***********************************************************************
 *                                                                     *
 *   DATE: 15 Feb 1990                                                 *
 *   VERSION: 1.0                                                      *
 *                                                                     *
 *   NAME: check_button_for_activity()                                 *
 *   MODULE NUMBER:                                                    *
 *   DESCRIPTION:                                                      *
 *         This module provides a means of checking the button clicked*
 *         by user mouse if the clicked button is right or left.       *
 *   ALGORITHM:                                                        *
 *   PASSED VARIABLES: window, event, arg(Sun variable)                *
 *                                                                     *
 *   RETURNS: None                                                     *
 *   GLOBAL VARIABLES USED: window, event, arg                         *
 *   GLOVAL VARIABLES CHANGED: None                                    *
 *   FILES READ: None                                                  *
 *   FILES WRITTEN:  None                                              *
 *   HARDWARE INPUT: None                                              *
 *   HARDWARE OUTPUT: None                                             *
 *   MODULES CALLED: event_id(), find_clicked_box(), event_x(),        *
 *                   event_y(), my_window_set(), null_proc(),          *
 *                   put_message()                                     *
 *   CALLING MODULES: check_activity()                                 *
 *                                                                     *
 *   AUTHOR: Intaek Kim                                                *
 *   HISTORY:                                                          *
 *   ABSTRACT DATA TYPE :                                              *
 *   ORDER OF :                                                        *
 **********************************************************************/
 void
 check_button_for_activity(window,event,arg)
  Window window;
  Event  *event;
  caddr_t arg;
  {
   extern my_window_set();
   extern put_message(), null_proc();

   /* Check for left or right button */
   switch(event_id(event)) {

     case MS_LEFT:
       if(event_is_up(event))
          find_clicked_box(event_x(event),event_y(event));
```

```
        break;

   case MS_RIGHT:
     my_window_set(null_proc);
     put_message(1,"ABORT -- Make another selection.");
     break;

   default:
     break;
   }
 return;
}
```

```
/**********************************************************************
 *                                                                    *
 *   DATE: 15 Feb 1990                                                *
 *   VERSION: 1.0                                                     *
 *                                                                    *
 *   NAME: check_activity()                                           *
 *   MODULE NUMBER:                                                   *
 *   DESCRIPTION:                                                     *
 *         This module provides a means of finally producing a file   *
 *         contained a set of predicate forms for an activity box.    *
 *   ALGORITHM:                                                       *
 *   PASSED VARIABLES: None                                           *
 *                                                                    *
 *   RETURNS: None                                                    *
 *   GLOBAL VARIABLES USED:  header_rootnode, box_rootnode            *
 *   GLOVAL VARIABLES CHANGED: None                                   *
 *   FILES READ: None                                                 *
 *   FILES WRITTEN: None                                              *
 *   HARDWARE INPUT: None                                             *
 *   HARDWARE OUTPUT: None                                            *
 *   MODULES CALLED: put_message(), strcmp(), my_move_cursor(),       *
 *                   my_window_set()                                  *
 *   CALLING MODULES: make_windows()                                  *
 *                                                                    *
 *   AUTHOR: Intaek Kim                                               *
 *   HISTORY:                                                         *
 *   ABSTRACT DATA TYPE :                                             *
 *   ORDER OF :                                                       *
**********************************************************************/
 void
 check_activity()
   {
     extern put_message(), my_window_set(), my_move_cursor();
     extern struct box_struct *box_rootnode;
     extern struct header_struct *header_rootnode;

     if(box_rootnode == NULL) {
        put_message(1,"FATAL: Can't check this empty diagram
       -- Make another selection.");
        return;
        }
     if(strcmp(header_rootnode->title.text_string,"") == 0) {
       put_message(1,"NO TITLE: Please enter the TITLE
      using EDIT DGM and then RETRY!!!");
     return;
```

```
  }
  my_move_cursor(INIT_LOC_X,INIT_LOC_Y);
  my_window_set(check_button_for_activity);
  put_message(1,"Move cursor inside activity box
 and click left button - Right to ABORT.");
  return;
}
```

```
/***********************************************************************
 *                                                                     *
 *   DATE: 15 Feb 1990                                                 *
 *   VERSION: 1.0                                                      *
 *                                                                     *
 *   NAME: create_temp_boundary_info()                                 *
 *   MODULE NUMBER:                                                    *
 *   DESCRIPTION:                                                      *
 *         This module provides a means of saving all information of   *
 *         boundary arrows of the IDEF0 diagram into the tempory       *
 *         file CHECKBOUNDARY.PRO.                                     *
 *   ALGORITHM:                                                        *
 *   PASSED VARIABLES: parentfile                                      *
 *                                                                     *
 *   RETURNS: None                                                     *
 *   GLOBAL VARIABLES USED: header_rootnode                            *
 *   GLOVAL VARIABLES CHANGED: None                                    *
 *   FILES READ: parentfile                                            *
 *   FILES WRITTEN: CHECKBOUNDARY.PRO                                  *
 *   HARDWARE INPUT: None                                              *
 *   HARDWARE OUTPUT: None                                             *
 *   MODULES CALLED: fopen(), put_message(), disable_input_window()    *
 *                   getc(), putc(), fclose(), search_boundary_info(),*
 *                   save_null_boundary(), printf()                    *
 *   CALLING MODULES: check_parent_box_file()                          *
 *                                                                     *
 *   AUTHOR: Intaek Kim                                                *
 *   HISTORY:                                                          *
 *   ABSTRACT DATA TYPE :                                              *
 *   ORDER OF :                                                        *
***********************************************************************/
 void
 create_temp_boundary_info(parentfile)
  char parentfile[];
   {
    extern put_message(),disable_input_window();
    FILE *parentfp, *childfp;
    extern struct header_struct *header_rootnode;
    extern int number_of_boundary_input,number_of_boundary_output;
    extern int number_of_boundary_control,number_of_boundary_mechanism;
    int ch;

    number_of_boundary_input = 0;
    number_of_boundary_output = 0;
    number_of_boundary_control = 0;
 number_of_boundary_mechanism = 0;
 if((parentfp = fopen(parentfile,"r")) == NULL ||
    (childfp = fopen("CHECKBOUNDARY.PRO","w")) == NULL) {
    put_message(1,"Unable to open the predicate file(s) -- ABORT");
    disable_input_window();
    }
 else {
    disable_input_window();
    while((ch = getc(parentfp)) != EOF)
      putc(ch,childfp);
    fclose(parentfp);
    search_boundary_info(childfp);
    save_null_boundary(childfp);
    if(fclose(childfp) != 0) printf("FILE CLOSE FAILED\n");
    }
 return;
}
```

```
/**********************************************************************
 *                                                                    *
 *   DATE: 15 Feb 1990                                                *
 *   VERSION: 1.0                                                     *
 *                                                                    *
 *   NAME: check_parent_box_file()                                    *
 *   MODULE NUMBER:                                                   *
 *   DESCRIPTION:                                                     *
 *         This module provides a means of checking if there is the   *
 *         file which the user specifies in the current directory.    *
 *   ALGORITHM:                                                       *
 *   PASSED VARIABLES: None                                           *
 *                                                                    *
 *   RETURNS: None                                                    *
 *   GLOBAL VARIABLES USED: None                                      *
 *   GLOVAL VARIABLES CHANGED: None                                   *
 *   FILES READ: None                                                 *
 *   FILES WRITTEN: None                                              *
 *   HARDWARE INPUT: None                                             *
 *   HARDWARE OUTPUT: None                                            *
 *   MODULES CALLED: strcpy(), panel_get_value(), fix_input(),        *
 *                   strcmp(), put_message(), disable_input_window(), *
 *                   strcat(), check_filename(),                      *
 *                   create_temp_boundary_info()                      *
 *   CALLING MODULES: check_button_for_boundary()                     *
 *                                                                    *
 *   AUTHOR: Intaek Kim                                               *
 *   HISTORY:                                                         *
 *   ABSTRACT DATA TYPE :                                             *
 *   ORDER OF :                                                       *
 **********************************************************************/
 check_parent_box_file()
  {
   extern put_message(), disable_input_window(),check_filename();
   extern my_move_cursor(),my_window_set(),fix_input();
   char name[FILE_NAME_LENGTH + 1],name2[FILE_NAME_LENGTH + 5];
   int  file_type_indicator;

   strcpy(name,(char *)panel_get_value(input_item));
   fix_input(name);
   if(strcmp(name,"") ==0) {
     put_message(1,"ABORT: No file name received
    --Make another selection.");
     disable_input_window();
     return(PANEL_NONE);
   }
 strcpy(name2,name);
 strcat(name2,".pro");
 file_type_indicator = check_filename(name2);
 switch (file_type_indicator) {

  case -1:
    disable_input_window();
    put_message(1,"ABORT: File does not exist
   --Make another selection.");
    break;

  case -3:
    disable_input_window();
    put_message(1,"ABORT: File is a directory
   -- Make another selection.");
    break;

  case -2: /* READ ONLY */
  case  0: /* READ/WRITE..  */
    put_message(1,"Enter prolog environment using another window.");
    create_temp_boundary_info(name2);
    break;

  default:
    put_message(1,"Unknown condition -- Make another selection.");
    disable_input_window();
    break;
  }
 return(PANEL_NONE);
}
```

```
/***********************************************************************
 *                                                                     *
 *   DATE: 15 Feb 1990                                                 *
 *   VERSION: 1.0                                                      *
 *                                                                     *
 *   NAME: check_button_for_boundary()                                 *
 *   MODULE NUMBER:                                                    *
 *   DESCRIPTION:                                                      *
 *         This module provides a means of conforming if the user      *
 *         would like to be continue to check IDEFO syntax for the     *
 *         boundary arrows in any IDEFO diagram.                       *
 *   ALGORITHM:                                                        *
 *   PASSED VARIABLES: window, event, arg(Sun variables)               *
 *                                                                     *
 *   RETURNS: None                                                     *
 *   GLOBAL VARIABLES USED: None                                       *
 *   GLOVAL VARIABLES CHANGED: None                                    *
 *   FILES READ: None                                                  *
 *   FILES WRITTEN: None                                               *
 *   HARDWARE INPUT: None                                              *
 *   HARDWARE OUTPUT: None                                             *
 *   MODULES CALLED: event_is_up(), event_id(), enable_input_window(),*
 *                   panel_set(), check_parent_box_file(),             *
 *                   put_message(), my_window_set(), null_proc()       *
 *   CALLING MODULES: check_boundary()                                 *
 *                                                                     *
 *   AUTHOR: Intaek Kim                                                *
 *   HISTORY:                                                          *
 *   ABSTRACT DATA TYPE :                                              *
 *   ORDER OF :                                                        *
 **********************************************************************/
void
check_button_for_boundary(window,event,arg)
  Window  window;
  Event   *event;
  caddr_t arg;
 {
  extern put_message(),enable_input_window(),my_window_set();
  extern null_proc();

  if(event_is_up(event)) return;

  switch event_id(event)
   {
     case MS_LEFT:
          enable_input_window();
          panel_set(input_item,
                    PANEL_VALUE_STORED_LENGTH,FILE_NAME_LENGTH,
                    PANEL_NOTIFY_PROC, check_parent_box_file,
                    0);
          put_message(1,"Enter the predicate file NAME
         with parent box and hit <Return>.");
          break;

        case MS_RIGHT:
          my_window_set(null_proc);
          put_message(1,"OPERATION ABORTED -- Make another selection.");
          break;

        default: break;
      }
     return;
    }
```

```
/**********************************************************************
 *                                                                    *
 *   DATE: 15 Feb 1990                                                *
 *   VERSION: 1.0                                                     *
 *                                                                    *
 *   NAME: check_boundary()                                           *
 *   MODULE NUMBER:                                                   *
 *   DESCRIPTION:                                                     *
 *         This module provides a means of producing all information *
 *         of a set of predicate forms of the boundary arrows in the  *
 *         IDEF0 diagram so as to check IDEF0 syntax of boundary      *
 *         arrows.                                                    *
 *   ALGORITHM:                                                       *
 *   PASSED VARIABLES: None                                           *
 *                                                                    *
 *   RETURNS: None                                                    *
 *   GLOBAL VARIABLES USED:  header_rootnode,line_rootnode            *
 *   GLOVAL VARIABLES CHANGED: None                                   *
 *   FILES READ: None                                                 *
 *   FILES WRITTEN: None                                              *
 *   HARDWARE INPUT: None                                             *
 *   HARDWARE OUTPUT: None                                            *
 *   MODULES CALLED: disable_input_window(), put_message(), strcmp(), *
 *                   my_move_cursor(), my_window_set()                *
 *   CALLING MODULES: make_windows()                                  *
 *                                                                    *
 *   AUTHOR: Intaek Kim                                               *
 *   HISTORY:                                                         *
 *   ABSTRACT DATA TYPE :                                             *
 *   ORDER OF :                                                       *
 *********************************************************************/
 void
 check_boundary()
  {
   extern put_message(),my_window_set();
   extern my_move_cursor(),disable_input_window();
   extern struct box_struct *box_rootnode;
   extern struct line_struct *line_rootnode;
   extern struct header_struct *header_rootnode;

   if(box_rootnode == NULL) {
     disable_input_window();
     put_message(1,"FATAL: Can't check this empty diagram
    -- Make another selection.");
     return;
    }
  if(line_rootnode == NULL) {
    put_message(1,"FATAL: Can't check this diagram
   -- Make another selection.");
    disable_input_window();
    return;
    }
  if(strcmp(header_rootnode->title.text_string,"") == 0) {
    put_message(1,"NO TITLE: Please enter the TITLE using EDIT DGM
   and then RETRY!!!");
    disable_input_window();
    return;
    }
  put_message(1,"CHECK BOUNDARY ARROW : |L| to check - |R| to ABORT");
  my_move_cursor(INIT_LOC_X,INIT_LOC_Y);
  my_window_set(check_button_for_boundary);
  return;
}
```

```
/**********************************************************************
 *                                                                    *
 *   DATE: 15 Feb 1990                                                *
 *   VERSION: 1.0                                                     *
 *                                                                    *
 *   NAME: save_null_boundary()                                       *
 *   MODULE NUMBER:                                                   *
 *   DESCRIPTION:                                                     *
 *         This module provides a means of saving the information     *
 *         which contains that there is no boundary arrow in          *
 *         accordance with ICOM.                                      *
 *   ALGORITHM:                                                       *
 *   PASSED VARIABLES: fp(file pointer)                               *
 *                                                                    *
 *   RETURNS: None                                                    *
 *   GLOBAL VARIABLES USED: number_of_boundary_input,                 *
 *                          number_of_boundary_output,                *
 *                          number_of_boundary_control,               *
 *                          number_of_boundary_mechanism              *
 *   GLOVAL VARIABLES CHANGED: None                                   *
 *   FILES READ: None                                                 *
 *   FILES WRITTEN: fp                                                *
 *   HARDWARE INPUT: None                                             *
 *   HARDWARE OUTPUT: None                                            *
 *   MODULES CALLED: fprintf(), itoa()                                *
 *   CALLING MODULES: create_temp_boundary_info()                     *
 *                                                                    *
 *   AUTHOR: Intaek Kim                                               *
 *   HISTORY:                                                         *
 *   ABSTRACT DATA TYPE :                                             *
 *   ORDER OF :                                                       *
 *********************************************************************/
save_null_boundary(fp)
 FILE   *fp;
 {
  extern int number_of_boundary_input,number_of_boundary_output;
  extern int number_of_boundary_control,number_of_boundary_mechanism;
  extern itoa();
  char buf[DESCRIPTION_LINE_LENGTH+1];

  if (number_of_boundary_input == 0)
     fprintf(fp,"confirmed([boundary_input, is, null]).\n");
  if (number_of_boundary_output == 0)
     fprintf(fp,"confirmed([boundary_output, is, null]).\n");
  if (number_of_boundary_control == 0)
```

```
    fprintf(fp,"confirmed([boundary_control, is, null]).\n");
 if (number_of_boundary_mechanism == 0)
    fprintf(fp,"confirmed([boundary_mechanism, is, null]).\n");
 itoa(number_of_boundary_input,buf);
 fprintf(fp,"confirmed([boundary_input, has_number, %s]).\n",buf);
 itoa(number_of_boundary_output,buf);
 fprintf(fp,"confirmed([boundary_output, has_number, %s]).\n",buf);
 itoa(number_of_boundary_control,buf);
 fprintf(fp,"confirmed([boundary_control, has_number, %s]).\n",buf);
 itoa(number_of_boundary_mechanism,buf);
 fprintf(fp,"confirmed([boundary_mechanism, has_number, %s]).\n",buf);
 return(MYTRUE);
}
```

```
/**********************************************************************
 *                                                                    *
 *   DATE: 15 Feb 1990                                                *
 *   VERSION: 1.0                                                     *
 *                                                                    *
 *   NAME: search_NLR_boundary_line_info()                            *
 *   MODULE NUMBER:                                                   *
 *   DESCRIPTION:                                                     *
 *              This module provides a means of working through the   *
 *              tree of line structure in left and right direction.   *
 *   ALGORITHM:                                                       *
 *   PASSED VARIABLES: line_info, fp                                  *
 *                                                                    *
 *   RETURNS: None                                                    *
 *   GLOBAL VARIABLES USED: None                                      *
 *   GLOVAL VARIABLES CHANGED: None                                   *
 *   FILES READ: None                                                 *
 *   FILES WRITTEN: None                                              *
 *   HARDWARE INPUT: None                                             *
 *   HARDWARE OUTPUT: None                                            *
 *   MODULES CALLED: save_boundary_line_info(),                       *
 *                   search_NLR_boundary_line_info()                  *
 *   CALLING MODULES: search_boundary_info()                          *
 *                                                                    *
 *   AUTHOR: Intaek Kim                                               *
 *   HISTORY:                                                         *
 *   ABSTRACT DATA TYPE :                                             *
 *   ORDER OF :                                                       *
 *********************************************************************/
  void
  search_NLR_boundary_line_info(line_info,fp)
  struct line_struct *line_info;
   FILE   *fp;
  {
   extern int number_of_boundary_input,number_of_boundary_output;
   extern int number_of_boundary_control,number_of_boundary_mechanism;

   if(line_info == NULL) return;
   else
     {
      save_boundary_line_info(line_info,fp);
      search_NLR_boundary_line_info(line_info->left,fp);
      search_NLR_boundary_line_info(line_info->right,fp);
      }
   return;
}
```

```
/*********************************************************************
 *                                                                   *
 *   DATE: 15 Feb 1990                                               *
 *   VERSION: 1.0                                                    *
 *                                                                   *
 *   NAME: search_boundary_info()                                    *
 *   MODULE NUMBER:                                                  *
 *   DESCRIPTION:                                                    *
 *              This module provides a means of working through the  *
 *              tree of the line structure in next direction.        *
 *   ALGORITHM:                                                      *
 *   PASSED VARIABLES: fp                                            *
 *                                                                   *
 *   RETURNS: None                                                   *
 *   GLOBAL VARIABLES USED: line_rootnode                            *
 *   GLOVAL VARIABLES CHANGED: None                                  *
 *   FILES READ: None                                                *
 *   FILES WRITTEN: None                                             *
 *   HARDWARE INPUT: None                                            *
 *   HARDWARE OUTPUT: None                                           *
 *   MODULES CALLED: fprintf(), search_NLR_boundary_line_info()      *
 *   CALLING MODULES: create_temp_boundary_info()                    *
 *                                                                   *
 *   AUTHOR: Intaek Kim                                              *
 *   HISTORY:                                                        *
 *   ABSTRACT DATA TYPE :                                            *
 *   ORDER OF :                                                      *
 *********************************************************************/
search_boundary_info(fp)
  FILE    *fp;
  {
   extern struct line_struct *line_rootnode;
   struct line_struct *line_info;
   extern int number_of_boundary_input,number_of_boundary_output;
   extern int number_of_boundary_control,number_of_boundary_mechanism;

   fprintf(fp,"confirmed([child_title, is, '%s']).\n",
           header_rootnode->title.text_string);
   line_info = line_rootnode;
   while(line_info != NULL)
     {
      search_NLR_boundary_line_info(line_info,fp);
      line_info = line_info->next;
      }
   return(MYTRUE);
}
```

```
/*********************************************************************
 *                                                                   *
 *   DATE : 4 Feb 1990                                               *
 *   VERSION : 1.0                                                   *
 *                                                                   *
 *   NAME : save_boundary_line_info()                                *
 *   MODULE NUMBER :                                                 *
 *   DESCRIPTION :                                                   *
 *      This module is to save the information of the boundary       *
 *      arrows in the IDEFO diagram.                                 *
 *   ALGORITHM :                                                     *
 *   PASSED VARIABLES : line_info, fp (File Pointer)                 *
 *                                                                   *
 *   RETURNS : None                                                  *
 *   GLOBAL VARIABLES USED : None                                    *
 *   GLOBAL VARIABLES CHANGED : None                                 *
 *   FILES READ : None                                               *
 *   FILES WRITTEN : fp                                              *
 *   HARDWARE INPUT : None                                           *
 *   HARDWARE OUTPUT : None                                          *
 *   MODULES CALLED : itoa(), fprintf()                              *
 *   CALLING MODULES : search_NLR_boundary_line_info()               *
 *                                                                   *
 *   AUTHOR : Intaek Kim                                             *
 *   HISTORY :                                                       *
 *   ABSTRACT DATA TYPE:                                             *
 *   ORDER OF:                                                       *
 ********************************************************************/

  save_boundary_line_info(line_info,fp)
   struct line_struct *line_info;
   FILE    *fp;
   {
    extern itoa();
    extern int number_of_boundary_input,number_of_boundary_output;
    extern int number_of_boundary_control,number_of_boundary_mechanism;
    char buf[DESCRIPTION_LINE_LENGTH+1];

    switch(line_info->start_ICOM[0])
      {
       case 'I':
        number_of_boundary_input += 1;
        itoa(number_of_boundary_input,buf);
        fprintf(fp,"confirmed([boundary_input%s, is, '%s']).\n",
                buf,line_info->label.text_string);
   break;

  case 'C':
   number_of_boundary_control += 1;
   itoa(number_of_boundary_control,buf);
   fprintf(fp, "confirmed([boundary_control%s, is, '%s']).\n",
           buf,line_info->label.text_string);
   break;

  case 'M':
   number_of_boundary_mechanism += 1;
   itoa(number_of_boundary_mechanism,buf);
   fprintf(fp,"confirmed([boundary_mechanism%s, is, '%s']).\n",
           buf ,line_info->label.text_string);
   break;

  default:
   if (line_info->end_ICOM[0] == 'O')
      {
       number_of_boundary_output += 1;
       itoa(number_of_boundary_output,buf);
       fprintf(fp,"confirmed([boundary_output%s, is, '%s']).\n",
               buf,line_info->label.text_string);
       break;
       }
  }
 return(MYTRUE);
}
```

```
/**********************************************************************
 *                                                                    *
 *   DATE : 2 Feb 1990                                                *
 *   VERSION : 1.0                                                    *
 *                                                                    *
 *   NAME :    save_header_info()                                     *
 *   MODULE NUMBER :                                                  *
 *   DESCRIPTION :                                                    *
 *        This module is to save the information of "NODE" and        *
 *        "TITLE" in the IDEFO diagram.                               *
 *                                                                    *
 *   ALGORITHM :                                                      *
 *   PASSED VARIABLES : fp (File pointer)                             *
 *                                                                    *
 *   RETURNS : None                                                   *
 *   GLOBAL VARIABLES USED : header_rootnode                          *
 *   GLOBAL VARIABLES CHANGED : None                                  *
 *   FILES READ : None                                                *
 *   FILES WRITTEN : fp                                               *
 *   HARDWARE INPUT : None                                            *
 *   HARDWARE OUTPUT : None                                           *
 *   MODULES CALLED : fprintf()                                       *
 *   CALLING MODULES :  store_predicates()                            *
 *                                                                    *
 *   AUTHOR : Intaek Kim                                              *
 *   HISTORY :                                                        *
 *   ABSTRACT DATA TYPE:                                              *
 *   ORDER OF:                                                        *
 **********************************************************************/
 save_header_info(fp)
   FILE     *fp;
   {
    extern put_message(),disable_input_window();
    extern struct header_struct *header_rootnode;

    fprintf(fp,"confirmed([title, is, '%s']).\n",
            header_rootnode->title.text_string);
    fprintf(fp,"confirmed([node, is, '%s']).\n",
            header_rootnode->node.text_string);
    return(MYTRUE);
   }
```

```
/**********************************************************************
 *                                                                    *
 *   DATE: 20 Feb 1990                                                *
 *   VERSIOIN: 1.0                                                    *
 *                                                                    *
 *   NAME:             traverse_boxes()                               *
 *   MODULE NUMBER:                                                   *
 *   DESCRIPTION:                                                     *
 *              This module provides a means of traversing all the    *
 *              activity boxes in a diagram and of passing the        *
 *              information of each box to store_diagram().           *
 *                                                                    *
 *   ALGORITHM:                                                       *
 *   PASSED VARIABLES:        fp                                      *
 *                                                                    *
 *   RETURNS: None                                                    *
 *   GLOBAL VARIABLES USED:   box_rootnode                            *
 *   GLOBAL VARIABLES CHANGED: None                                   *
 *   FILES READ: None                                                 *
 *   FILES WRITTEN: None                                              *
 *   HARDWARE INPUT: None                                             *
 *   HARDWARE OUTPUT: None                                            *
 *   MODULES CALLED:  save_arrow_info_of_abox()                       *
 *   CALLING MODULES: store_predicates()                              *
 *                                                                    *
 *   AUTHOR:  Intaek Kim                                              *
 *   HISTORY:                                                         *
 *   ABSTRACT DATA TYPE:                                              *
 *   ORDER OF:                                                        *
**********************************************************************/
  void
  traverse_boxes(fp)
   FILE    *fp;
   {
    extern struct box_struct *box_rootnode;
    struct box_struct *temp_box;

    temp_box = box_rootnode;
    while(temp_box != NULL)
     {
      save_arrow_info_of_abox(fp,temp_box);
      temp_box = temp_box->next;
      }
    return;
   }
```

```
/**********************************************************************
*                                                                     *
*   DATE: 15 Feb 1990                                                 *
*   VERSION: 1.0                                                      *
*                                                                     *
*   NAME: store_predicates()                                          *
*   MODULE NUMBER:                                                    *
*   DESCRIPTION:                                                      *
*         This module provides a means of saving all information of  *
*         arrows(ICOM) attatched on the activity boxes in the IDEFO  *
*         diagram.                                                    *
*   ALGORITHM:                                                        *
*   PASSED VARIABLES: file_name                                       *
*                                                                     *
*   RETURNS: None                                                     *
*   GLOBAL VARIABLES USED:  None                                      *
*   GLOVAL VARIABLES CHANGED: None                                    *
*   FILES READ: None                                                  *
*   FILES WRITTEN: None                                               *
*   HARDWARE INPUT: None                                              *
*   HARDWARE OUTPUT: None                                             *
*   MODULES CALLED: fopen(), put_message(), disable_input_window(),  *
*                   save_header_info(), traverse_boxes(), fclose()   *
*   CALLING MODULES: overwrite_predictes()                            *
*                                                                     *
*   AUTHOR: Intaek Kim                                                *
*   HISTORY:                                                          *
*   ABSTRACT DATA TYPE :                                              *
*   ORDER OF :                                                        *
*****  ***************************************************************/
 void
 store_predicates(file_name)
   char file_name[];
   {
    extern put_message(),disable_input_window();
    FILE   *fp;

    if ((fp = fopen(file_name,"w")) == NULL) {
        put_message(1,"Unable to open the file for predicates
                        -- ABORT.");
        disable_input_window();
        }
    else {
          disable_input_window();
          save_header_info(fp);
```

```
       traverse_boxes(fp);
       if(fclose(fp) != 0) printf("FILE CLOSE FAILED\n");
       }
 return;
}
```

```
/**********************************************************************
 *                                                                    *
 *   DATE: 15 Feb 1990                                                *
 *   VERSION: 1.0                                                     *
 *                                                                    *
 *   NAME: overwrite_predicates()                                     *
 *   MODULE NUMBER:                                                   *
 *   DESCRIPTION:                                                     *
 *         This module provides a means of overwriting the predicate  *
 *         forms into the existing flie which is specified by the user*
 *   ALGORITHM:                                                       *
 *   PASSED VARIABLES: window, event, arg(Sun variables)              *
 *                                                                    *
 *   RETURNS: None                                                    *
 *   GLOBAL VARIABLES USED: last_file_name                            *
 *   GLOVAL VARIABLES CHANGED: None                                   *
 *   FILES READ: None                                                 *
 *   FILES WRITTEN:  file pointer passed to a .pro file               *
 *   HARDWARE INPUT: None                                             *
 *   HARDWARE OUTPUT: None                                            *
 *   MODULES CALLED: event_is_up(), event_id(), strcpy(),             *
 *                   store_predicates(), put_message(), null_proc(),  *
 *                   my_window_set()                                  *
 *   CALLING MODULES: get_filename_for_predicates()                   *
 *                                                                    *
 *   AUTHOR: Intaek Kim                                               *
 *   HISTORY:                                                         *
 *   ABSTRACT DATA TYPE :                                             *
 *   ORDER OF :                                                       *
 **********************************************************************/
  void
  overwrite_predicates(window,event,arg)
    Window window;
    Event  *event;
    caddr_t arg;
    {
     extern null_proc(), put_message(),my_window_set();
     char file_name[FILE_NAME_LENGTH + 5];

     if (!event_is_up(event)) return;
     switch(event_id(event))
      {
       case MS_LEFT:
         strcpy(file_name,last_file_name);
         store_predicates(file_name);
```

```
   put_message(1,"OVERWRITE DONE -- Make another selection.");
   my_window_set(null_proc);
   break;

 case MS_RIGHT:
   my_window_set(null_proc);
   put_message(1,"ABORT overwrite -- Make another selection.");
   break;
   }
 return;
}
```

```
/*********************************************************************
*                                                                    *
*   DATE:    10  Jan 1990                                            *
*   VERSION: 1.0                                                     *
*                                                                    *
*   NAME: get_filename_for_predicates()                              *
*   MODULE NUMBER:                                                   *
*   DESCRIPTION:                                                     *
*       The purpose of this module is to get the file name from the  *
*       user in which to save the predicates file.                   *
*   ALGORITHM:                                                       *
*   PASSED VARIABLES: None                                           *
*   RETURNS:        PANEL_NONE (Sunview variable)                    *
*   GLOBAL VARIABLES USED: None                                      *
*   GLOBAL VARIABLES CHANGED: None                                   *
*   FILES READ: None                                                 *
*   FILES WRITTEN: None                                              *
*   HARDWARE INPUT: None                                             *
*   HARDWARE OUTPUT: None                                            *
*   MODULES CALLED: put_message(),fix_input(),disable_input_window(),*
*                   check_filename(),my_window_set(),stordicates(),  *
*                   my_move_cursor();                                *
*   CALLING MODULES: save_predicates()                               *
*                                                                    *
*   AUTHOR:  Intaek Kim                                              *
*   HISTORY:                                                         *
*   ABSTRACT DATA TYPE:                                              *
*   ORDER OF:                                                        *
*********************************************************************/
Panel_setting
 get_filename_for_predicates()
  {
   extern put_message(),disable_input_window();
   extern fix_input(), my_window_set(),my_move_cursor();
   extern check_filename();
   char name[FILE_NAME_LENGTH+1],name2[FILE_NAME_LENGTH+5];
   int file_type_indicator;

   /* get the user input(file name) */
   strcpy(name,(char *)panel_get_value(input_item));
   fix_input(name); /* Remove blanks and replace \n to \0 */

   if(strcmp(name,"")==0)
    {
     put_message(1,"OPERATION ABORTED -- NO FILE NAME RECEIVED
  -- Make another selection");
   disable_input_window();
   return(PANEL_NONE);
  }

strcpy(name2,name);
strcat(name2,".pro");
/* Checks file name of "name2" is what type of file. */
file_type_indicator = check_filename(name2);

switch(file_type_indicator)
 {
  case -1:
   store_predicates(name2);
   put_message(1,"SAVE DONE -- Make another selection.");
   break;
  case -2:
   disable_input_window();
   put_message(1,"Can't overwrite file -- READ ONLY
 -- Make another selection.");
   break;
  case -3:
   disable_input_window();
   put_message(1,"File is a DIRECTORY -- Make another selection.");
   break;
  case 0:
   put_message(1,"FILE EXISTS - |L| to overwrite, - |R| to ABORT.");
   my_move_cursor(INIT_LOC_X,INIT_LOC_Y);
   strcpy(last_file_name,name2);
   my_window_set(overwrite_predicates);
   break;
  default:
   put_message(1,"Unknown condition -- Make another selection");
   disable_input_window();
   break;
   }
 return(PANEL_NONE);
}
```

```
/***********************************************************************
 *                                                                     *
 *   DATE:    10 Jan 1990                                              *
 *   VERSION: 1.0                                                      *
 *                                                                     *
 *   NAME: save_predicates()                                           *
 *   MODULE NUMBER:                                                    *
 *   DESCRIPTION:                                                      *
 *       This module provides a means of asking user for continuing to *
 *   save a predicate file(.pro) or to abort this function.            *
 *                                                                     *
 *   ALGORITHM:                                                        *
 *   PASSED VARIABLES:                                                 *
 *                                                                     *
 *   GLOBAL VARIABLES USED: *box_rootnode, *header_rootnode            *
 *   GLOBAL VARIABLES CHANGED:None                                     *
 *   FILES READ:                                                       *
 *   FILES WRITTEN:                                                    *
 *   HARDWARE INPUT:                                                   *
 *   HARDWARE OUTPUT:                                                  *
 *   MODULES CALLED: get_filename_for_predicates()                     *
 *   CALLING MODULES: make_windows()                                   *
 *                                                                     *
 *   AUTHOR:  Intaek Kim                                               *
 *   HISTORY :                                                         *
 *   ABSTRACT DATA TYPE:                                               *
 *   ORDER OF:                                                         *
***********************************************************************/
 void
 save_predicates()
  {
   extern  put_message(), enable_input_window();
   extern  disable_input_window();
   extern  struct box_struct *box_rootnode;
   extern  struct header_struct *header_rootnode;

   if(box_rootnode == NULL) {
     put_message(1,"FATAL: Can't save this empty diagram
    -- Make another selection.");
     disable_input_window();
     return;
     }
   if(strcmp(header_rootnode->title.text_string,"") == 0) {
     disable_input_window();
     put_message(1,"NO TITLE: Please enter the TITLE
 using EDIT DGM and then RETRY!!");
  return;
  }
enable_input_window();
panel_set(input_item,
          PANEL_VALUE_STORED_LENGTH,FILE_NAME_LENGTH,
          PANEL_NOTIFY_PROC, get_filename_for_predicates,
          0);
put_message(1,"Enter the file name and hit <Return>.");
return;
}
```

### *IDEF$_0$ Syntax Expert System*

This section presents the source code documentations for the IDEF$_0$ Syntax Expert System and contains the inference engine(ISES) and two rule bases: Activity IDEF$_0$ Syntax Rules and Boundary IDEF$_0$ Syntax Rules.

```
/*****************************************************************
 *                                                               *
 * DATE: 25 Feb. 1990                                            *
 * VERSION: 1.1                                                  *
 * NAME: BC3                                                     *
 * DESCRIPTION: The purpose of this module is to provide an      *
 *              inference engine for checking IDEFO syntax.      *
 *              BC3 provides a means of a shell for backward     *
 *              chaining control strategy.                       *
 * OPERATING SYSTEM: UNIX 4.3                                    *
 * LANGUAGE: Quintus Prolog                                      *
 * CONTENTS: *                                                   *
 * AUTHOR: DR. Frank M. Brown                                    *
 * HISTORY: Version 1.0 - MS-DOS version(DR. Frank M. Brown)     *
 *          Version 1.1 - UNIX 4.3 version(Intaek Kim)           *
 *****************************************************************

/*****************************************************************/
/*                                                               */
/*                              BC3                              */
/*                                                               */
/*       A shell for backward-chaining expert systems.           */
/*                                                               */
/* Each item of knowledge is represented by a triple, i.e.,      */
/* a three-element list of the form [Object,Attribute,Value].    */
/*                                                               */
/* An associated rule-base supplies the following data:          */
/*                                                               */
/* 1. A goals-statement, in the form of a list of triples to     */
/*    be solved in sequence. The solved triples are printed      */
/*    by the shell.                                              */
/* 2. A collection of if-then rules for triples.                 */
/* 3. A collection of 'fact' triples, i.e., triples asserted     */
/*    as known a priori.                                         */
/* 4. A collection of 'askable' triples, indicating the forms    */
/*    of triples whose values may be obtained from the user.     */
/* 5. A collection of 'keep' triples, indicating the form of     */
/*    the triples not to be erased from working memory at        */
/*    the beginning of a new session.                            */
/*                                                               */
/* Each item of knowledge stored in working memory is of the     */
/* form confirmed([Obj,Attr,Val]) or denied([Obj,Attr,Val]).     */
/*                                                               */
/* To use the system, load BC3, load the appropriate rule-       */
/* base and type 'start.' Because BC3's operator-defini-      */
/* tions are used by the rule-bases, BC3 must load first.     */
/*                                                            */
/**************************************************************/

/*------------------- OPERATOR DEFINITIONS -------------------*/
/*                                                            */
/* The operators defined below enable the rules in the know-  */
/* ledge-base to be expressed in a form more readable than    */
/* the standard (prefix) form.                                */
/*                                                            */
/*------------------------------------------------------------*/

?- op(250, xfx, ::).
?- op(245, xfx, then).
?- op(240, fx,  if).
?- op(235, xfx, derived_from).
?- op(230, xfy, or).
?- op(225, xfy, and).
?- op(220,  fy, not).

/*------------------------- START ----------------------------*/
/*                                                            */
/* The procedure 'start' begins by erasing from working mem-  */
/* ory all 'confirmed' and 'denied' clauses, except those     */
/* clauses protected by 'keep' from erasure. The list of      */
/* goal-triples is then read from the rule-base and solved in */
/* turn by 'solve'. A trace is maintained of the back-        */
/* chaining search-tree generated in solving the goals. When  */
/* the last of the goal-triples is solved, the values of all  */
/* goals, except those solved by asking the user directly,    */
/* are displayed; the trace is also displayed, if requested,  */
/* as a "how" explanation of the solution.                    */
/*                                                            */
/*------------------------------------------------------------*/

erase_working_memory :-
  ( confirmed(Triple),              /* Erase all working-mem- */
    not(keep::Triple),              /* ory elements not pro-  */
    retract(confirmed(Triple))      /* tected by 'keep' state-*/
  ;                                 /* ments in the knowledge-*/
    denied(Triple),                 /* base.                  */
    not(keep::Triple),
    retract(denied(Triple)) ),
  fail.
```

```
erase_working_memory.

start :-
  ask_about_verbose,
  fail.

start :-
  ask_about_checking_type,
  fail.

start :-
  retractall(why_trace(_)),          /* Erase the "why" trace. */
  goals:: Goals,                     /* Find the goal-triples, */
  prefix(Goals,PrefixedGoals),       /* prefix each of them    */
  show_head_message,                 /* with the word 'goal',  */
  solve(Goals,[],PartTrace),         /* satisfy all of the     */
  !,nl,                              /* goals and then put the */
  append(PrefixedGoals,PartTrace,Trace),
                                     /* list of goals at the   */
                                     /* front of the "how"     */
  ask_about_trace(Trace),            /* trace. Supply a "how"  */
  ask_about_saving_working_memory,   /* explanation on request.*/
  erase_working_memory,
  start_message.

start :-                             /* If all triples can't   */
  nl,                                /* be solved, announce it.*/
  write('I can''t solve this problem.'),nl,
  start_message.

/*--------------------------- SOLVE ---------------------------*/
/*                                                             */
/*  The predicate 'solve(Goals,Trace,New_trace)' means that    */
/*  Goals is a list of goals (expressed as triples), and that  */
/*  Trace and New_trace are, respectively, the trace-lists be- */
/*  and after solution of the goal at the head of the goal-    */
/*  list. The procedure 'solve' solves each of the goals in    */
/*  turn. The first step in solving a goal is to erase the     */
/*  "why" trace and to initialize it with that goal. Thus each */
/*  goal is solved with a separate "why" trace. As each rule   */
/*  is encountered in descending through the search-tree for a */
/*  given goal, that rule is added to the front of the "why"   */
/*  trace.                                                     */
/*                                                             */
```

```
/*--------------------------------------------------------------*/

solve([],Trace,Trace).
solve([Goal|Others],Trace,NewTrace) :-
  retractall(why_trace(_)),           /* Initialize the "why"   */
  asserta(why_trace([goal::Goal])),   /* trace.                 */
  is_known(Goal,Trace,Trace1),
  ( confirmed(Goal),!                /* Write each triple as    */
  ;                                  /* it's solved, but don't */
    nl,write_triple(Goal),nl ),      /* write a triple that's  */
  solve(Others,Trace1,NewTrace).     /* been told explicitly   */
                                     /* by the user.           */

write_triple([Obj,Attr,Val]) :-
  writelist([Obj,' ',Attr,' ',Val,'.']).


/*------------------------- IS_KNOWN -------------------------*/
/*                                                            */
/*  The 'is_known' procedure maintains a trace of the path of */
/*  the solution-tree leading to the triple currently under   */
/*  consideration. 'is_known(Triple,Trace,NewTrace)' means    */
/*  that if reasoning to a certain point has been recorded in */
/*  the list 'Trace', then the additional triple 'Triple' is  */
/*  known via reasoning recorded by the list 'NewTrace'.      */
/*                                                            */
/*--------------------------------------------------------------*/

/* A triple is not known if it has been denied by the user.   */

is_known(Triple,Trace,Trace) :-
  denied(Triple),
  !,
  fail.

/* A triple is known if it is already logged in the trace.    */

is_known([O,A,V],Trace,[in_trace::[Tag,[O,A,V]]|Trace]) :-
  member(Tag::[O,A,V],Trace),
  Tag \== confirmed_not,
  !.

/* A triple is known if it has been confirmed by the user.    */

is_known(Triple,Trace,[was_told::Triple|Trace]) :-
```

```
  confirmed(Triple).

/* A triple is known if it is a fact in the rule-base.        */

is_known(Triple,Trace,[fact::Triple|Trace]) :-
  fact:: Triple.

/* A triple [X,P,Y] is known if the Prolog goal P(X,Y) suc-    */
/* ceeds, either because P is a built-in predicate, or because */
/* the rule-base has prolog-code defining P. The triple        */
/* [2,member,[1,2]], for example, is converted into the goal   */
/* member(2,[1,2]), which is then executed by Prolog. To keep  */
/* non-Prolog-programmers out of trouble, the triple [X,is,Y]  */
/* is trapped so that it will not be executed as an arithmetic */
/* statement. The triple [X,:=,Y] is interpreted as Prolog's   */
/* arithmetic or assignment goal, X is Y.                      */
/*                                                             */
/* This kind of triple, which runs off and does a computation, */
/* is called a "procedural attachment", or "demon."            */

is_known([Obj,Attr,Val],Trace,[solved::[Obj,Attr,Val]|Trace]) :-
  atom(Attr),              /*    Attr must be a legal functor.   */
  (
    Attr == :=, !,
    Obj is Val             /* Interpret ':=' as Prolog's 'is'.   */
  ;
    not (check_reserved_words(Attr)),
 /* Interpret everything else, except  */
    T =.. [Attr,Obj,Val],/* reserved words for rule-base, as a */
 /* functor on a two-place            */
    T, !                   /* predicate to be solved as a goal.  */
  ).

/* A triple is known if it is the head of a rule and the con-  */
/* ditions of the rule are satisfied. We put a rule that we    */
/* encounter at the head of the "why" trace, erasing any du-   */
/* plicates of the rule that are already in the "why" trace.   */
/* The "why" trace is maintained in the database, in a clause  */
/* of the form 'why_trace(<List of goals and rules>)'. This    */
/* differs from the "how" trace, which is handed as an argu-   */
/* ment from goal to goal.                                     */

is_known(Triple,Trace,[was_proved::[Triple,Rule]|Trace]) :-
  member(Rule:: Triple derived_from _Conds,Trace),
  !.
```

```
is_known(Trpl,Trc,[Rule:: Trpl derived_from Conds|Trc1]) :-
  Rule:: if Conds then Trpl,
  ( verbose,
    writelist(['Trying ', Rule, ':: ', Trpl]),nl
  ;
    not verbose),
  why_trace(WhyTrace),
  remove(Rule:: Trpl derived_from Conds,WhyTrace,PartWhy),
  append([Rule:: Trpl derived_from Conds],PartWhy,NewWhy),
  retract(why_trace(_)),
  asserta(why_trace(NewWhy)),
  is_known(Conds,Trc,Trc1),
  ( verbose,
    writelist(['Proved ', Rule, ':: ', Trpl]),nl
  ;
    not verbose ),
  !.

/* A condition involving "and", "or", or "not" is known if its */
/* parts are known in suitable combinations.                   */

is_known(Triples1 and Triples2,Trace,Trace2) :-
  is_known(Triples1,Trace,Trace1),
  is_known(Triples2,Trace1,Trace2).

is_known(Triples1 or _Triples2,Trace,Trace1) :-
  is_known(Triples1,Trace,Trace1).
is_known(_Triples1 or Triples2,Trace,Trace2) :-
  is_known(Triples2,Trace,Trace2).

is_known(not Triple,Trace,[confirmed_not::Triple|Trace]) :-
  not is_known(Triple,Trace,_Trace1).

/* A triple is known if (a) the rule-base classifies it as     */
/* "askable" and if (b) the user confirms it. The user may     */
/* request a "why" explanation before responding to the ques-  */
/* tion.                                                       */

is_known([O,A,V],Trace,[was_told::[O,A,V]|Trace]):-
  askable:: [O,A,_],    /* 'ask_about' causes the side-effect   */
  ask_about([O,A,V]),  /* of confirming or denying [O,A,V] in  */
  !,                   /* working memory. The clause succeeds  */
  confirmed([O,A,V]).  /* if the triple was confirmed.         */
```

```
/*------------------------ ASK_ABOUT ------------------------*/

/* If the user is asked about a triple [O,A,V] in which V is   */
/* a variable, we assume that only one value of V is allowed   */
/* for that triple. The askable-fact in the rule-base is to    */
/* have the form 'askable::[O,A,LegalVals]',where LegalVals is */
/* either a string describing legal values or a list enumer-   */
/* ating such values. When the user supplies a legal value for */
/* V, the triple is confirmed in working memory.               */

ask_about([Obj,Attr,Val]) :-
  var(Val),
  !,
  not confirmed([Obj,Attr,_]),
  nl, writelist([Obj,' ',Attr,'? ']),nl,
  askable:: [Obj,Attr,LegalValues],
  write('Legal values: '), write(LegalValues), nl,
  write('> '), read(Reply),
  ( /* If the user replies 'why.', give him an explanation and */
    /* ask again for a value.                                  */
    (
      means(Reply,why),
      explain_why([Obj,Attr,Val]),
      !,
      ask_about([Obj,Attr,Val])
    )
  ;
    /* If LegalValues is a list, check that the reply is in    */
    /* the list.                                               */
    (
      atom(LegalValues)            /* LegalValues is a string.*/
    ;
      LegalValues = [_|_],         /* LegalValues is a list.  */
      member(Reply,LegalValues)
    ),
    !,
    assertz(confirmed([Obj,Attr,Reply]))
  ;
    write('Please re-enter your reply.'),nl,
    ask_about([Obj,Attr,Val])
  ).

/* If we get to this clause, the user is being asked to reply  */
/* yes or no concerning a triple [O,A,V] in which V is not a   */
/* variable. For given O and V, working memory may store more  */
/* than one triple, confirmed or denied, having different val- */
/* ues of V.                                                   */

ask_about([Obj,Attr,Val]) :-
  not confirmed([Obj,Attr,Val]),
  not denied([Obj,Attr,Val]),
  nl,
  writelist([Obj,' ',Attr,' ',Val,'? (yes./no./why.)']),
  nl,write('> '),read(Reply),
  (
    means(Reply,yes),
    assertz(confirmed([Obj,Attr,Val])), !
  ;
    means(Reply,no),
    assertz(denied([Obj,Attr,Val])), !
  ;
    means(Reply,why),
    explain_why([Obj,Attr,Val]),
    !,
    ask_about([Obj,Attr,Val])
  ;
    write('Please re-enter your reply.'),nl,
    ask_about([Obj,Attr,Val])
  ).

/*-------------------- ASK ABOUT VERBOSE -----------------*/

ask_about_verbose :-
  retractall(verbose),
  write(' Question: Do you want verbose operation(y./n.)? '),
  !,
  read(Reply),nl,nl,
  means(Reply,yes),
  assert(verbose).

/*-------------- ASK ABOUT CHECKING TYPE ------------------*/

ask_about_checking_type :-
  write(' Question:
  Do you wish to check ACTIVITY BOX, BOUNDARY ARROWS  '),nl,
  write('          or to have HELP MESSAGES ?'),nl,nl,
  write('        To check ACTIVITY BOX    -> Enter a.'),nl,
  write('        To check BOUNDARY ARROWS -> Enter b.'),nl,
  write('        To have HELP MESSAGE     -> Enter h.'),nl,
  write('Choice : '),
  read(Reply),nl,
  ( ( Reply == a,
      load_sarule('ACTIVITYSARULE.PRO'),
      load_working_memory('CHECKBOX.PRO'),
      ! )
   ;
    ( Reply == b,
      load_sarule('BOUNDARYSARULE.PRO'),
      load_working_memory('CHECKBOUNDARY.PRO'),
      ! )
   ;
    ( Reply == h,
      help_messages,
      !,
      ask_about_checking_type )
   ;
    ( write(' Please re-enter your choice!!!!'),nl,nl,
      ask_about_checking_type )
  ).

/*------------------- ASK ABOUT TRACE ---------------------*/

ask_about_trace(Trace) :-
  nl,nl,
  write(' Question: Do you wish to see how this answer '), nl,
  write('          was arrived at(y./n.)? '),
  read(Reply),
  ( means(Reply,yes), !,
    write_trace(Trace)
  ;
    true ).

/*-------------- ASK ABOUT SAVING WORKING MEMORY --------------*/

ask_about_saving_working_memory :- nl,
  write('/*****************!!! WORNING !!!************************/'),nl,
  write('/* After this session, all working memory elements will */'),nl,
  write('/* be erased except for elements being protected by     */'),nl,
  write('/*   keep   statements in the knowledge base.           */'),nl,
  write('/********************************************************/'),nl,
  nl,
  write(' Question: Do you wish to save the current working memory'),nl,
  write('in a file(y./n.)? '),
  read(Reply),nl,
  ( means(Reply,yes),
    save_working_memory,
    erase_working_memory, !
  ;
    erase_working_memory ).

/*----------------------- EXPLAIN_WHY -----------------------*/

explain_why(Triple) :-
  why_trace(WhyTrace),
  write('Because::'),nl,
  justify(Triple,WhyTrace).

justify(Triple,WhyTrace) :-
  member(goal::Goal,WhyTrace),
  Triple = Goal,
  writelist(['This will satisfy the goal ',Goal]),nl,
  nl,
  !.
justify(Triple,WhyTrace) :-
  member(Rule::Head derived_from Cs,WhyTrace),
  ( among(Triple,Cs),
    writelist(['I can use ',Triple]),nl
  ;
    among(not Triple,Cs),
    writelist(['I can use NOT ',Triple]),nl
  ),
  remove(Rule::Head derived_from Cs,WhyTrace,NewTrace),
  list_known_triples(Cs),
  writelist(['     to help satisfy ',Rule,':: ',Head]),nl,nl,
  justify(Head,NewTrace).

list_known_triples(Cs) :-
  among(Triple,Cs),
  (
    confirmed(Triple)
  ;
    fact:: Triple
  ),
  writelist(['     knowing ',Triple]),nl,
  fail.
list_known_triples(_).

among(Triple,Conditions) :-
  Triple = Conditions.
```

```
among(Triple, FirstTriple and OtherConditions) :-
  Triple = FirstTriple
  ;
  among(Triple,OtherConditions).
among(Triple, FirstTriple or OtherConditions) :-
  Triple = FirstTriple
  ;
  among(Triple,OtherConditions).

why :-                    /* Diagnostic utilities for          */
  why_trace(Trace),       /* the why-trace.                    */
  write_trace(Trace).

list_why :-
  why_trace(Trace),
  member(M,Trace),
  write(M),nl,nl,
  fail.

why_candidates :-
  why_trace(Trace),
  member(_Rule:: Head derived_from Cs,Trace),
  among(Triple,Cs),
  write('Head   = '),write(Head),nl,
  write('Triple = '),write(Triple),nl,nl,
  fail.


/*------------------------ WRITE_TRACE ------------------------*/

write_trace([]) :-
  nl.
write_trace([Tag::[O,A,V]|Rest]) :-
  ( Tag == goal,     !, write('GOAL::   ')
  ;
    Tag == fact,     !, write('FACT::   ')
  ;
    Tag == solved,   !, write('SOLVED:: ')
  ;
    Tag == was_told, !, write('TOLD::   ')
  ;
    Tag == confirmed_not, !, write('CONTRADICTED:: ')
  ),
  write([O,A,V]),nl,
  write_trace(Rest).
write_trace([in_trace::[Tag,Triple]|Rest]) :-
  !,
  write('KNOWN::  '),write(Tag),write(':: '),write(Triple),nl,
  write_trace(Rest).
write_trace([was_proved::[Triple,Rule]|Rest]) :- !,
  write('PROVED:: '),write(Triple),write(' using '),write(Rule),nl,
  write_trace(Rest).
write_trace([Rule:: Triple derived_from Conditions|Rest]) :- !,
  writelist([Rule,':: ',Triple,'  Was Derived From']),nl,
  write_conditions(Conditions),
  write_trace(Rest).
write_trace([X|Rest]) :-
  write(X),nl,
  write_trace(Rest).

write_conditions([X,Y,Z]) :-
  tab(8),write([X,Y,Z]),nl.
write_conditions(not [X,Y,Z]) :-
  tab(4),write('NOT '),write([X,Y,Z]),nl.
write_conditions([X,Y,Z] and Conditions) :- !,
  tab(8),write([X,Y,Z]),write(' AND'),nl,
  write_conditions(Conditions).
write_conditions(not [X,Y,Z] and Conditions) :- !,
  tab(4),write('NOT '),write([X,Y,Z]),write(' AND'),nl,
  write_conditions(Conditions).
write_conditions(Conditions1 and Conditions2) :-
  write_conditions(Conditions1),tab(8),write('AND'),nl,
  write_conditions(Conditions2).
write_conditions([X,Y,Z] or Conditions) :- !,
  tab(8),write([X,Y,Z]),write(' OR'),nl,
  write_conditions(Conditions).
write_conditions(Conditions1 or Conditions2) :-
  write_conditions(Conditions1),tab(8),write('OR'),nl,
  write_conditions(Conditions2).
write_conditions(not [X,Y,Z] or Conditions) :-
  tab(4),write('NOT '),write([X,Y,Z]),write(' OR'),nl,
  write_conditions(Conditions).


/*------------------------ FILE I/O -------------------------*/

get_filename(Filename) :-
  write('Please supply a filename: '),
  read(Filename).
```

```
load_sarule(SAruleType) :-
  retractall(_::_),
  see(SAruleType),
  load_file,
  seen.

load_file :-
  read(Term),
  load(Term).

load(end_of_file) :- !.
load(Term) :-
  assertz(Term),
  load_file.

load_working_memory(CheckFileType) :-
  retractall(confirmed(_)),
  retractall(denied(_)),
  see(CheckFileType),
  load_file,
  seen.

save_working_memory :-
  get_filename(Filename),
  tell(Filename),
  save_wme,
  told.

save_wme :-
  confirmed(Triple),
  writeq(confirmed(Triple)),
  write('.'),nl,
  fail.

save_wme :-
  denied(Triple),
  writeq(denied(Triple)),
  write('.'),nl,
  fail.

save_wme.

/*-------------------- UTILITY PROCEDURES ---------------------*/

check_reserved_words(Attr) :-
```

```
    member(Attr,[' ',is,input_is,has_input_number,output_is,
 has_output_number,control_is,has_control_number,
 mechanism_is,has_mechanism_number,number_is,
 should_be,has_number]).

show_head_message :-
  nl,nl,
  write('  /*************** IDEFO Syntax Messages ***************/'),
  nl.

help_messages:-
            reconsult('HELP.PRO').

not(Predicate) :-
   call(Predicate), !, fail.
not(_).

writelist([]).
writelist([X|L]) :-
  write(X),
  writelist(L).

member(X,[X|_]).
member(X,[_|L]) :-
  member(X,L).

append([],L,L).
append([X|L],M,[X|N]) :-
  append(L,M,N).

remove(_,[],[]).
remove(X,[X|L],M) :-
  !,
  remove(X,L,M).
remove(X,[Y|L],[Y|M]) :-
  remove(X,L,M).

prefix([],[]).
prefix([Goal|Goals],[goal::Goal|PrefixedGoals]) :-
  prefix(Goals,PrefixedGoals).

list_working_memory :-
  confirmed(Triple),
  write(confirmed(Triple)),write('.'),nl,
  fail.
```

```
list_working_memory :-
  denied(Triple),
  write(denied(Triple)),
  write('.'),nl,
  fail.
list_working_memory.

means(Reply,yes) :-
  member(Reply,[y,yes]).
means(Reply,no) :-
  member(Reply,[n,no]).
means(Reply,why) :-
  member(Reply,[why,w]).

start_message :-
   write('/********************************************************/'),nl,
   write('/*                                                      */'),nl,
   write('/*     WELCOME TO IDEFO SYNTAX EXPERT SYSTEMS           */'),nl,
   write('/*                                                      */'),nl,
   write('/* I.Type  start.  to begin a new session.              */'),nl,
   write('/*                                                      */'),nl,
   write('/* II. Answer all questions using lower case,ending with*/'),nl,
   write('/*     a period.                                        */'),nl,
   write('/*                                                      */'),nl,
   write('/* III. Type    halt.    to exit prolog session.        */'),nl,
   write('/*                                                      */'),nl,
   write('/********************************************************/'),nl,
   nl.

?- start_message.
```

## *Activity $IDEF_0$ Syntax Rules*

```
/******************************************************************
 *                                                                *
 * DATE: 25 Apr.  1990                                            *
 * VERSION: 1.0                                                   *
 * NAME: ACTIVITYSARULE.PRO                                       *
 * TITLE: Rule base for an activity box                           *
 * COORDINATOR: Intaek Kim                                        *
 * PROJECT: Knowledge base                                        *
 * OPERATING SYSTEM: UNIX 4.3                                     *
 * LANGUAGE: Quintus Prolog                                       *
 * FILE PROCESSING: This module should be used with an inference  *
 *                  engine, BC3.                                  *
 * CONTENTS: Rules for checking IDEF0 syntax of an activity box.  *
 * HISTORY:                                                       *
 ******************************************************************

/********************* GOALS *************************/
/* These lists of goal present the resulting message. */

  goals:: [['Name', ' ', _Name],      /* Goal for NAME of the box */
   ['Input', ' ', _Input],     /* Goal for INPUT of box    */
           ['Output', ' ', _Output],  /* Goal for OUTPUT of box   */
           ['Control', ' ', _Control],/* Goal for CONTROL of box  */
           ['Mechanism', ' ', _Mechanism], /* Goal for MECHANISM  */
           ['Number', ' ', _Number]]. /* Goal for the box NUMBER  */


/********************** RULES ******************************/
/**************** About Activity Name *********************/
 rule1 :: if [activityname, is, '']
  then ['Name', ' ', '      --> ERROR: No Activity Name.
     Each box must have an activity name'].

 rule2 :: if [activityname, is, Activity]
     and [Activity, \==, '']
          then ['Name', ' ', '    --> CORRECT: Activity Name is OK'].

/******************** About Input *************************/
 rule3 :: if [activityname, is, Activity]
            and [Activity, input_is, null]
          then ['Input', ' ', '--> CORRECT: No Input Arrows, however,
                 Input is OK'].

 rule4 :: if [activityname, is, Activity]
```

```
            and [Activity, input_is, '']
          then ['Input', ' ', '    --> ERROR: No Input Label
 Each Input arrow must have an input label'].

 rule5 :: if [activityname, is, Activity]
    and [Activity, has_input_number, InputNumber]
            and [InputNumber, >, 5]
         then ['Input', ' ', '    --> RECOMMEND:
                You would better reduce the number of Input arrows
                from 0 to 5'].

 rule6 :: if [activityname, is, _Activity]
         then ['Input', ' ', '    --> CORRECT: Input is OK'].

/******************** About Output **********************/
/* If there is a box and the output of the box is empty, */
/* then there is no output/name.                         */

 rule7 :: if [activityname, is, Activity]
             and [Activity, output_is, null]
      then ['Output', ' ', '   --> ERROR: You should have at least
             one output arrow'].

 rule8 :: if [activityname, is, Activity]
             and [Activity, output_is, '']
         then ['Output', ' ', '    --> ERROR: No Output Label.
                Each Output Arrow should have an output Label'].

 rule9 :: if [activityname, is, Activity]
            and [Activity, has_output_number, OutputNumber]
    and [OutputNumber, >, 5]
         then ['Output', ' ', '    --> RECOMMEND:
        You would better reduce the number of Output arrows
                from 1 to 5'].

 rule10 ::  if [activityname, is, _Activity]
             then ['Output', ' ', '    --> CORRECT: Output is OK'].

/******************* About Control *******************/
 rule11 :: if [activityname, is, Activity]
             and [Activity, control_is, null]
    then ['Control', ' ', '  --> ERROR: You should have at least
           one control arrow'].

 rule12 :: if [activityname, is, Activity]
```

```
             and [Activity, control_is, '']
          then ['Control', ' ', '  --> ERROR: No Control Label.
                 Each Control Arrow should have a control Label'].

 rule13 :: if [activityname, is, Activity]
            and [Activity, has_control_number, ControlNumber]
    and [ControlNumber, >, 5]
         then ['Control', ' ', '  --> RECOMMEND:
      You would better reduce the number of Control arrows
              from 1 to 5'].

 rule14 :: if [activityname, is, _Activity]
          then ['Control', ' ', '  --> CORRECT: Control is OK'].

/****************** About Mechanism *****************/
 rule15 :: if [activityname, is, Activity]
             and [Activity, mechanism_is, '']
          then ['Mechanism', ' ', '--> ERROR: No Mechanism Label.
              Each Mechanism Arrow should have a mechanism Label'].

 rule16 :: if [activityname, is, Activity]
             and [Activity, mechanism_is, null]
   then ['Mechanism', ' ', '--> CORRECT: No Mechanism Arrows, however,
          Mechanism is OK'].

 rule17 :: if [activityname, is, Activity]
    and [Activity, has_mechanism_number, MechanismNumber]
            and [MechanismNumber, >, 5]
         then ['Mechanism', ' ', '--> RECOMMEND:
                You would better reduce the number of Mechanism arrows
        within 0 to 5'].

 rule18 :: if [activityname, is, _Activity]
          then ['Mechanism', ' ', '--> CORRECT: Mechanism is OK'].

/********************** Activity Number *********************/
/* If there is a box and the number of the box is 0, that is, */
/* the box has no number, the box must be the top most box.   */

 rule19 :: if [title, is, Activity]
      and [Activity, number_is, Number]
      and [Number, ==, 0]
           then ['Number', ' ', '--> CORRECT: Activity number is OK.
  This activity must be the top most level box'].
 rule20 :: if [title, is, Activity]
      and [Activity, number_is, Number]
      and [Number, \==, 0]
           then ['Number', ' ', '--> ERROR: This activity must be
  the top most level box.
  This activity don''t need a box number'].

/* Each activity box must have a number for the box within */
/* 1 through 6 except for the top most level activity box. */

/* If there is a box and the number of the box is 0, then */
/* there is no number in the box.                         */

 rule21 :: if [Activity, number_is, Number]
               and [title, is, ParentActivity]
       and [ParentActivity, \==, Activity]
       and [Number, ==, 0]
           then ['Number', ' ', '--> ERROR: Activity box has no number.
         The activity box should have a box number from 1 to 6.
                 (If the activity box is the top most level one, then
                  ignore this message)'].

 rule22 :: if [Activity, number_is, _Number]
       and [title, is, ParentActivity]
       and [ParentActivity, \==, Activity]
       and [activity_number, is, in_legal_range]
            then ['Number', ' ', '--> CORRECT: Activity box number
   is OK.'].

 rule23 :: if [Activity, number_is, _Number]
       and [title, is, ParentActivity]
       and [ParentActivity, \==, Activity]
       and [activity_number, is, in_illegal_range]
         then ['Number', ' ', '   --> ERROR: Activity box number is
not proper.
        The recommended range of number is 1 to 6'].

/* These rules are the second level rules.   */
 rule24 :: if [_Activity, number_is, Number]
       and [Number, >, 0]
               and [Number, <, 7]
            then [activity_number, is, in_legal_range].

 rule25 :: if [_Activity, number_is, Number]
      and [Number, <, 0]
```

```
          then [activity_number, is, in_illegal_range].

rule26 :: if [_Activity, number_is, Number]
             and [Number, >, 6]
          then [activity_number, is, in_illegal_range].
```

*Boundary $IDEF_0$ Syntax Rules*

```
/********************************************************************
 *                                                                  *
 * DATE: 25 Apr. 1990                                               *
 * VERSION: 1.0                                                     *
 * FILE NAME: BOUNDARYSARULE.PRO                                    *
 * TITLE: Rule base for boundary arrows                             *
 * COORDINATOR: Intaek Kim                                          *
 * PROJECT: Knowledge base                                          *
 * OPERATING SYSTEM: UNIX 4.3                                       *
 * LANGUAGE: Quintus Prolog                                         *
 * FILE PROCESSING: This module should be used an inference engine  *
 *                  BC3.                                            *
 * CONTENTS: Rules for checking IDEFO syntax of boundary arrows.    *
 * HISTORY:                                                         *
 ********************************************************************

/********************* GOALS ***************************/
/* These lists of goal present the resulting message. */

 goals:: [['Boundary Input', ' ', _Input],
   /* Goal for boundary input in any IDEFO diagram */
          ['Boundary Output', ' ', _Output],
   /* Goal for boundary output in any IDEFO diagram */
          ['Boundary Control', ' ', _Control],
   /* Goal for boundary control in any IDEFO diagram */
          ['Boundary Mechanism', ' ', _Mechanism]
   /* Goal for boundary mechanism in any IDEFO diagram */
  ].

/********************** RULES ***********************/
/**** When there is no information of parent box ***/
 rule1 :: if [child_title, is, ParentBox]
             and not [activityname, is, ParentBox]
          then [boundarysarule, is, stalled].

 rule2 :: if  [boundarysarule, is, stalled]
          then ['Boundary Input', ' ', '
 --> !!! THIS IS A FATAL ERROR !!!'].

 rule3 :: if [boundarysarule, is, stalled]
          then ['Boundary Output', ' ', '
 --> There is nothing about Parent activity'].

 rule4 :: if [boundarysarule, is, stalled]
```

```
          then ['Boundary Control', ' ', '
 --> Maybe you have tried to check syntax with
         a file without PARENT ACTIVITY BOX information'].

 rule5 :: if [boundarysarule, is, stalled]
          then ['Boundary Mechanism', ' ', '
 --> PLEASE START AGAIN !!!'].

/******************* About Boundary Input ******************/
/* No label on boundary input arrow */
 rule6 :: if [boundary_input, is, '']
          then ['Boundary Input', ' ','
 --> ERROR: No boundary input label'].

/* No label on input arrow of the parent box */
 rule7 :: if [child_title, is, ParentBox]
            and [ParentBox, input_is, '']
         then ['Boundary Input',' ','
--> ERROR: Parent Input has no label'].

/*   The number of Input arrow(s) of the Parent Activity  */
/* box is differ from that of the Boundary Input arrow(s).*/
/*   The number of the Parent Input arrow(s) > The number */
/* of the Boundary Input arrow(s).                        */
 rule8 :: if [child_title, is, ParentBox]
            and [ParentBox, has_input_number, InputNumber]
            and [boundary_input, has_number, BoundInNumber]
            and [InputNumber, >, BoundInNumber]
         then ['Boundary Input',' ','
--> ERROR: The number of Input arrow(s) of
                Parent Activity box is greater than that of
         Boundary Input arrow(s) -- Must have the same number'].

/*   The number of Input arrow(s) of the Parent Activity box is */
/* differ from that of the Boundary Input arrow(s).             */
/* The number of the Parent Input arrow(s) < The number of the  */
/* Boundary Input arrow(s).                                     */
 rule9 :: if [child_title, is, ParentBox]
            and [ParentBox, has_input_number, InputNumber]
            and [boundary_input, has_number, BoundInNumber]
            and [InputNumber, <, BoundInNumber]
         then ['Boundary Input',' ','
--> ERROR: The number of Input arrow(s) of
                Parent Activity box is less than that of
         Boundary Input arrow(s) -- Must have the same number'].
```

```
/*  If the number of arrows is greater than five, recomend  */
/* about how many the number of arrows exists.              */
 rule10 :: if [boundary_input, has_number, BoundInNumber]
             and [child_title, is, ParentBox]
             and [ParentBox, has_input_number, BoundInNumber]
             and [BoundInNumber, >, 5]
          then ['Boundary Input', ' ', '    --> RECOMMEND:
          You would better reduce the number of arrows to six
          below'].

/* No boundary Input arrow and No Input at Parent Box          */
 rule11 :: if [child_title, is, ParentBox]
             and [boundary_input, is, null]
             and [ParentBox, input_is, null]
          then ['Boundary Input', ' ','
--> CORRECT: Boundary Input is OK'].

/*  Consider the correct case acording to the Number of */
/* Input arrow(s).                                      */
 rule12 :: if [boundary_input, has_number, BoundInNumber]
              and [child_title, is, ParentBox]
              and [ParentBox, has_input_number, BoundInNumber]
          then [case_of_boundary_in, is, BoundInNumber].

 /* Case 1: The number of Boundary Input arrow is 1.  */
 rule13 :: if [case_of_boundary_in, is, 1]
              and [child_title, is, ParentBox]
              and [ParentBox, input_is, ParentInput]
              and [boundary_input1, is, ParentInput]
          then ['Boundary Input', ' ','
--> CORRECT: Boundary Input is OK'].

 /* Case 2: The number of Boundary Input arrows is 2.  */
 rule14 :: if [case_of_boundary_in, is, 2]
              and [boundary_input1, is, ParentInput1]
              and [boundary_input2, is, ParentInput2]
              and [child_title, is, ParentBox]
              and [ParentBox, input_is, ParentInput1]
              and [ParentBox, input_is, ParentInput2]
          then ['Boundary Input', ' ','
--> CORRECT: Boundary Input is OK'].

 /* Case 3: The number of Boundary Input arrows is 3.  */
 rule15 :: if [case_of_boundary_in, is, 3]
```

```
       and [boundary_input1, is, ParentInput1]
       and [boundary_input2, is, ParentInput2]
       and [boundary_input3, is, ParentInput3]
       and [child_title, is, ParentBox]
       and [ParentBox, input_is, ParentInput1]
       and [ParentBox, input_is, ParentInput2]
       and [ParentBox, input_is, ParentInput3]
     then ['Boundary Input', ' ','
--> CORRECT: Boundary Input is OK'].

/* Case 4: The number of Boundary Input arrows is 4.  */
rule16 :: if [case_of_boundary_in, is, 4]
       and [boundary_input1, is, ParentInput1]
       and [boundary_input2, is, ParentInput2]
       and [boundary_input3, is, ParentInput3]
       and [boundary_input4, is, ParentInput4]
       and [child_title, is, ParentBox]
       and [ParentBox, input_is, ParentInput1]
       and [ParentBox, input_is, ParentInput2]
       and [ParentBox, input_is, ParentInput3]
       and [ParentBox, input_is, ParentInput4]
     then ['Boundary Input', ' ','
--> CORRECT: Boundary Input is OK'].

/* Case 5: The number of Boundary Input arrows is 5.  */
rule17 :: if [case_of_boundary_in, is, 5]
       and [boundary_input1, is, ParentInput1]
       and [boundary_input2, is, ParentInput2]
       and [boundary_input3, is, ParentInput3]
       and [boundary_input4, is, ParentInput4]
       and [boundary_input5, is, ParentInput5]
       and [child_title, is, ParentBox]
       and [ParentBox, input_is, ParentInput1]
       and [ParentBox, input_is, ParentInput2]
       and [ParentBox, input_is, ParentInput3]
       and [ParentBox, input_is, ParentInput4]
       and [ParentBox, input_is, ParentInput5]
     then ['Boundary Input', ' ','    --> CORRECT: Boundary Input
is OK'].

/* Boundary Input is not matched */
/* This rule includes all No_match case though both of */
/* parent and child have the same number of input      */
/* arrow(s).                                           */
rule18 :: if [child_title, is, ParentBox]
```

```
       and [activityname, is, ParentBox]
     then ['Boundary Input',' ','--> ERROR: Boundary Input is not
     matched. -- You may have at least one unmatched label'].

/******************** About Boundary Output ************************/
 rule19 :: if [boundary_output, is, '']
    then ['Boundary Output', ' ',' --> ERROR: No boundary output
   label'].

 rule20 :: if [child_title, is, ParentBox]
       and [ParentBox, output_is, '']
    then ['Boundary Output',' ',' --> ERROR: Parent Output has no
   label'].

 rule21 :: if [boundary_output, is, null]
     then ['Boundary Output',' ',' --> ERROR: No boundary output
 arrow.
         Should have at least one boundary output arrow'].

 rule22 :: if [child_title, is, ParentBox]
       and [ParentBox, output_is, null]
     then ['Boundary Output', ' ', '
 --> ERROR: No parent output arrow.
         Should have at least one parent output arrow'].

/*  The number of Output arrow(s) of the Parent Activity  */
/* box is differ from that of the Boundary Output arrow(s).*/
/*  The number of the Parent Output arrow(s) > The number */
/* of the Boundary Output arrow(s).                       */
 rule23 :: if [child_title, is, ParentBox]
      and [ParentBox, has_output_number, OutputNumber]
      and [boundary_output, has_number, BoundOutNumber]
      and [OutputNumber, >, BoundOutNumber]
     then ['Boundary Output',' ',' --> ERROR: The number of Output
     arrow(s) of Parent Activity box is greater than that of
     Boundary Output arrow(s) -- Must have the same number'].

/*  The number of Output arrow(s) of the Parent Activity box is */
/* differ from that of the Boundary Output arrow(s).            */
/* The number of the Parent Output arrow(s) < The number of the */
/* Boundary Output arrow(s).                                    */
 rule24 :: if [child_title, is, ParentBox]
      and [ParentBox, has_output_number, OutputNumber]
      and [boundary_output, has_number, BoundOutNumber]
      and [OutputNumber, <, BoundOutNumber]
```

```
        then ['Boundary Output',' ','   --> ERROR: The number of Output
        arrow(s) of Parent Activity box is less than that of
        Boundary Output arrow(s) -- Must have the same number'].

/*   If the number of arrows is greater than five, recomend  */
/* about how many the number of arrows exists.              */
 rule25 :: if [boundary_output, has_number, BoundOutNumber]
           and [child_title, is, ParentBox]
           and [ParentBox, has_output_number, BoundOutNumber]
           and [BoundOutNumber, >, 5]
        then ['Boundary Output', ' ', '    --> RECOMMEND:
        You would better reduce the number of arrows to six
        below'].

/*  Consider the correct case acording to the Number of */
/* Output arrow(s).                                     */
 rule26 :: if [boundary_output, has_number, BoundOutNumber]
            and [child_title, is, ParentBox]
            and [ParentBox, has_output_number, BoundOutNumber]
        then [case_of_boundary_out, is, BoundOutNumber].

 /* Case 1: The number of Boundary Output arrow is 1.  */
 rule27 :: if [case_of_boundary_out, is, 1]
            and [boundary_output1, is, ParentOutput]
            and [child_title, is, ParentBox]
            and [ParentBox, output_is, ParentOutput]
        then ['Boundary Output', ' ','   --> CORRECT:
                        Boundary Output is OK'].

 /* Case 2: The number of Boundary Output arrows is 2.  */
 rule28 :: if [case_of_boundary_out, is, 2]
            and [boundary_output1, is, ParentOutput1]
            and [boundary_output2, is, ParentOutput2]
            and [child_title, is, ParentBox]
            and [ParentBox, output_is, ParentOutput1]
            and [ParentBox, output_is, ParentOutput2]
        then ['Boundary Output', ' ','   --> CORRECT:
                        Boundary Output is OK'].

 /* Case 3: The number of Boundary Output arrows is 3.  */
 rule29 :: if [case_of_boundary_out, is, 3]
            and [boundary_output1, is, ParentOutput1]
            and [boundary_output2, is, ParentOutput2]
            and [boundary_output3, is, ParentOutput3]
            and [child_title, is, ParentBox]
```

```
                and [ParentBox, output_is, ParentOutput1]
                and [ParentBox, output_is, ParentOutput2]
                and [ParentBox, output_is, ParentOutput3]
            then ['Boundary Output', ' ',' --> CORRECT:
                        Boundary Output is OK'].

/* Case 4: The number of Boundary Output arrows is 4.  */
rule30 :: if [case_of_boundary_out, is, 4]
                and [boundary_output1, is, ParentOutput1]
                and [boundary_output2, is, ParentOutput2]
                and [boundary_output3, is, ParentOutput3]
                and [boundary_output4, is, ParentOutput4]
                and [child_title, is, ParentBox]
                and [ParentBox, output_is, ParentOutput1]
                and [ParentBox, output_is, ParentOutput2]
                and [ParentBox, output_is, ParentOutput3]
                and [ParentBox, output_is, ParentOutput4]
            then ['Boundary Output', ' ',' --> CORRECT:
                        Boundary Output is OK'].

/* Case 5: The number of Boundary Output arrows is 5.  */
rule31 :: if [case_of_boundary_out, is, 5]
                and [boundary_output1, is, ParentOutput1]
                and [boundary_output2, is, ParentOutput2]
                and [boundary_output3, is, ParentOutput3]
                and [boundary_output4, is, ParentOutput4]
                and [boundary_output5, is, ParentOutput5]
                and [child_title, is, ParentBox]
                and [ParentBox, output_is, ParentOutput1]
                and [ParentBox, output_is, ParentOutput2]
                and [ParentBox, output_is, ParentOutput3]
                and [ParentBox, output_is, ParentOutput4]
                and [ParentBox, output_is, ParentOutput5]
            then ['Boundary Output', ' ',' --> CORRECT:
                        Boundary Output is OK'].

/* Boundary Output is not matched */
/* This rule includes all No_match case though both of */
/* parent and child have the same number of output      */
/* arrow(s).                                            */
rule32 :: if [child_title, is, ParentBox]
                 and [activityname, is, ParentBox]
             then ['Boundary Output',' ','--> ERROR: Boundary Output is not
             matched. -- You may have at least one unmatched label'].
```

```
/**************** About Boundary Control ******************/
 rule33 :: if [boundary_control, is, '']
       then ['Boundary Control', ' ','
      --> ERROR: No boundary control label'].

 rule34 :: if [child_title, is, ParentBox]
             and [ParentBox, control_is, '']
      then ['Boundary Control',' ','
     --> ERROR: Parent Control has no label'].

 rule35 :: if [boundary_control, is, null]
          then ['Boundary Control',' ','
 --> ERROR: No boundary control arrow.
                 Should have at least one boundary control arrow'].

 rule36 :: if [child_title, is, ParentBox]
             and [ParentBox, control_is, null]
          then ['Boundary Control', ' ', '
 --> ERROR: No parent control arrow.
                 Should have at least one parent control arrow'].

/*   If the number of arrows is greater than five, recomend  */
/* about how many the number of arrows exists.               */
 rule37 :: if [boundary_control, has_number, BoundConNumber]
            and [child_title, is, ParentBox]
            and [ParentBox, has_control_number, BoundConNumber]
            and [BoundConNumber, >, 5]
         then ['Boundary Control', ' ', '  --> RECOMMEND:
         You would better reduce the number of arrows to six
         below'].

/*  Consider the correct case acording to the Number of */
/* Control arrow(s).                                    */
 rule38 :: if [boundary_control, has_number, BoundConNumber]
             and [child_title, is, ParentBox]
             and [ParentBox, has_control_number, BoundConNumber]
         then [case_of_boundary_con, is, BoundConNumber].

 /* Case 1: The number of Boundary Control arrow is 1.  */
 rule39 :: if [case_of_boundary_con, is, 1]
             and [boundary_control1, is, ParentControl]
             and [child_title, is, ParentBox]
             and [ParentBox, control_is, ParentControl]
         then ['Boundary Control', ' ',' --> CORRECT: Boundary
                         Control is OK'].
```

```
/* Case 2: The number of Boundary Control arrows is 2.  */
rule40 :: if [case_of_boundary_con, is, 2]
            and [boundary_control1, is, ParentControl1]
            and [boundary_control2, is, ParentControl2]
            and [child_title, is, ParentBox]
            and [ParentBox, control_is, ParentControl1]
            and [ParentBox, control_is, ParentControl2]
        then ['Boundary Control', ' ',' --> CORRECT: Boundary
                        Control is OK'].

/* Case 3: The number of Boundary Control arrows is 3.  */
rule41 :: if [case_of_boundary_con, is, 3]
            and [boundary_control1, is, ParentControl1]
            and [boundary_control2, is, ParentControl2]
            and [boundary_control3, is, ParentControl3]
            and [child_title, is, ParentBox]
            and [ParentBox, control_is, ParentControl1]
            and [ParentBox, control_is, ParentControl2]
            and [ParentBox, control_is, ParentControl3]
        then ['Boundary Control', ' ',' --> CORRECT: Boundary
                        Control is OK'].

/* Case 4: The number of Boundary Control arrows is 4.  */
rule42 :: if [case_of_boundary_con, is, 4]
            and [boundary_control1, is, ParentControl1]
            and [boundary_control2, is, ParentControl2]
            and [boundary_control3, is, ParentControl3]
            and [boundary_control4, is, ParentControl4]
            and [child_title, is, ParentBox]
            and [ParentBox, control_is, ParentControl1]
            and [ParentBox, control_is, ParentControl2]
            and [ParentBox, control_is, ParentControl3]
            and [ParentBox, control_is, ParentControl4]
        then ['Boundary Control', ' ',' --> CORRECT: Boundary
                        Control is OK'].

/* Case 5: The number of Boundary Control arrows is 5.  */
rule43 :: if [case_of_boundary_con, is, 5]
            and [boundary_control1, is, ParentControl1]
            and [boundary_control2, is, ParentControl2]
            and [boundary_control3, is, ParentControl3]
            and [boundary_control4, is, ParentControl4]
            and [boundary_control5, is, ParentControl5]
            and [child_title, is, ParentBox]
```

```
             and [ParentBox, control_is, ParentControl1]
             and [ParentBox, control_is, ParentControl2]
             and [ParentBox, control_is, ParentControl3]
             and [ParentBox, control_is, ParentControl4]
             and [ParentBox, control_is, ParentControl5]
         then ['Boundary Control', ' ',' --> CORRECT: Boundary
                         Control is OK'].

 /* Boundary Control is not matched */
 /* This rule includes all No_match case though both of */
 /* parent and child have the same number of control    */
 /* arrow(s).                                           */
 rule44 :: if [child_title, is, ParentBox]
              and [activityname, is, ParentBox]
          then ['Boundary Control',' ','-> ERROR: Boundary Control is not
          matched. -- You may have at least one unmatched label'].

/**************** About Boundary Mechanism **************/
 rule45 :: if [boundary_mechanism, is, '']
     then ['Boundary Mechanism', ' ','
    --> ERROR: No boundary mechanism label'].

 rule46 :: if [child_title, is, ParentBox]
             and [ParentBox, mechanism_is, '']
     then ['Boundary Mechanism',' ','
    --> ERROR: Parent Mechanism has no label'].

/*   The number of Mechanism arrow(s) of the Parent Activity  */
/* box is differ from that of the Boundary Mechanism arrow(s).*/
/*   The number of the Parent Mechanism arrow(s) > The number */
/* of the Boundary Mechanism arrow(s).                        */
 rule47 :: if [child_title, is, ParentBox]
            and [ParentBox, has_mechanism_number, MecNumber]
            and [boundary_mechanism, has_number, BoundMecNumber]
            and [MecNumber, >, BoundMecNumber]
         then ['Boundary Mechanism',' ','--> ERROR: The number of
         Mechanism arrow(s) of Parent Activity box is greater than that
         of Boundary Mechanism arrow(s) -- Must have the same
         number'].

/*   The number of Mechanism arrow(s) of the Parent Activity box is */
/* differ from that of the Boundary Mechanism arrow(s).             */
/* The number of the Parent Mechanism arrow(s) < The number of the  */
/* Boundary Mechanism arrow(s).                                     */
 rule48 :: if [child_title, is, ParentBox]
```

```
            and [ParentBox, has_mechanism_number, MecNumber]
            and [boundary_mechanism, has_number, BoundMecNumber]
            and [MecNumber, <, BoundMecNumber]
         then ['Boundary Mechanism',' ','--> ERROR: The number of
         Mechanism arrow(s) of Parent Activity box is less than
         that of Boundary Mechanism arrow(s) -- Must have the same
         number'].

/*  If the number of arrows is greater than five, recomend  */
/* about how many the number of arrows exists.             */
 rule49 :: if [boundary_mechanism, has_number, BoundMecNumber]
            and [child_title, is, ParentBox]
            and [ParentBox, has_mechanism_number, BoundMecNumber]
            and [BoundMecNumber, >, 5]
         then ['Boundary Mechanism', ' ', '--> RECOMMEND:
         You would better reduce the number of arrows to six
         below'].

/* No boundary Mechanism arrow and No Mechanism at Parent Box */
 rule50 :: if [child_title, is, ParentBox]
            and [boundary_mechanism, is, null]
            and [ParentBox, mechanism_is, null]
         then ['Boundary Mechanism', ' ','--> CORRECT: Boundary
                          Mechanism is OK'].

/*  Consider the correct case acording to the Number of */
/* Mechanism arrow(s).                                  */
 rule51 :: if [boundary_mechanism, has_number, BoundMecNumber]
             and [child_title, is, ParentBox]
             and [ParentBox, has_mechanism_number, BoundMecNumber]
         then [case_of_boundary_mech, is, BoundMecNumber].

 /* Case 1: The number of Boundary Mechanism arrow is 1.  */
 rule52 :: if [case_of_boundary_mech, is, 1]
             and [child_title, is, ParentBox]
             and [ParentBox, mechanism_is, ParentMech]
             and [boundary_mechanism1, is, ParentMech]
         then ['Boundary Mechanism', ' ','--> CORRECT: Boundary
                          Mechanism is OK'].

 /* Case 2: The number of Boundary Mechanism arrows is 2.  */
 rule53 :: if [case_of_boundary_mech, is, 2]
             and [boundary_mechanism1, is, ParentMech1]
             and [boundary_mechanism2, is, ParentMech2]
             and [child_title, is, ParentBox]
            and [ParentBox, mechanism_is, ParentMech1]
            and [ParentBox, mechanism_is, ParentMech2]
        then ['Boundary Mechanism', ' ','--> CORRECT: Boundary
                       Mechanism is OK'].

/* Case 3: The number of Boundary Mechanism arrows is 3.  */
rule54 :: if [case_of_boundary_mech, is, 3]
            and [boundary_mechanism1, is, ParentMech1]
            and [boundary_mechanism2, is, ParentMech2]
            and [boundary_mechanism3, is, ParentMech3]
            and [child_title, is, ParentBox]
            and [ParentBox, mechanism_is, ParentMech1]
            and [ParentBox, mechanism_is, ParentMech2]
            and [ParentBox, mechanism_is, ParentMech3]
        then ['Boundary Mechanism', ' ','--> CORRECT: Boundary
                       Mechanism is OK'].

/* Case 4: The number of Boundary Mechanism arrows is 4.  */
rule55 :: if [case_of_boundary_mech, is, 4]
            and [boundary_mechanism1, is, ParentMech1]
            and [boundary_mechanism2, is, ParentMech2]
            and [boundary_mechanism3, is, ParentMech3]
            and [boundary_mechanism4, is, ParentMech4]
            and [child_title, is, ParentBox]
            and [ParentBox, mechanism_is, ParentMech1]
            and [ParentBox, mechanism_is, ParentMech2]
            and [ParentBox, mechanism_is, ParentMech3]
            and [ParentBox, mechanism_is, ParentMech4]
        then ['Boundary Mechanism', ' ','--> CORRECT: Boundary
                       Mechanism is OK'].

/* Case 5: The number of Boundary Mechanism arrows is 5.  */
rule56 :: if [case_of_boundary_mech, is, 5]
            and [boundary_mechanism1, is, ParentMech1]
            and [boundary_mechanism2, is, ParentMech2]
            and [boundary_mechanism3, is, ParentMech3]
            and [boundary_mechanism4, is, ParentMech4]
            and [boundary_mechanism5, is, ParentMech5]
            and [child_title, is, ParentBox]
            and [ParentBox, mechanism_is, ParentMech1]
            and [ParentBox, mechanism_is, ParentMech2]
            and [ParentBox, mechanism_is, ParentMech3]
            and [ParentBox, mechanism_is, ParentMech4]
            and [ParentBox, mechanism_is, ParentMech5]
        then ['Boundary Mechanism', ' ','--> CORRECT: Boundary
```

```
                             Mechanism is OK'].

/* Boundary Mechanism is not matched */
/* This rule includes all No_match case though both of */
/* parent and child have the same number of mechanism  */
/* arrow(s).                                           */
rule57 :: if [child_title, is, ParentBox]
             and [activityname, is, ParentBox]
         then ['Boundary Mechanism',' ','
--> ERROR: Boundary Mechanism is not
         matched. -- You may have at least one unmatched label'].
```

# *Bibliography*

1. Blackburn, Mark R. "Using Expert Systems to Construct Formal Specifications," *IEEE Expert. AI/Software Engineering*, Spring, 1989.

2. Blackman, Michael J. "CASE for Expert Systems," *AI Expert*, Vol.5, No.2:27-31, Feburary 1990.

3. Booch, Grady. *Software Engineering with Ada.* Benjamin and Cummings Publishing Company, Menlo Park CA, 1987.

4. Bratko, Ivan. *Prolog Programming for Artificial Intelligence.* England: Addison-Wesley Publishing Company, 1987.

5. Brown, Donald E. "Inference Engines for the Mainstream," *AI Expert*, Vol.5, No.2:32-37, Feburary 1990.

6. Brown, Frank M. *A Shell for Backward-Chaining Expert Systems.* Handout, Department of Electrical and Computer Engineering, Air Force Institute of Technology(AU), Wright-Patterson, OH, 1989.

7. Charniak, Eugene. and McDermott, Drew V. *Introduction to Artificial Intelligence.* Addison-Wesley Publishing Company, 1987.

8. Hartrum, Thomas C. *System Development Documentation Guidelines and Standards (Draft # 4).* Department of Electrical and Computer Engineering, Air Force Institute of Technology(AU), Wright-Patterson AFB, OH, January 1989.

9. Johnson, Steven E. *A Graphics Editor for Structured Analysis with a Data Dictionary.* MS thesis, AFIT/GE/ENG/87D-28, School of Engineering, Air Force Institute of Technology(AU), Wright-Patterson AFB, OH, December 1987.

10. Jung, Donghak *Design of a Syntax Validation Tool for Requirements Analysis Using Structured Analysis and Design Technique(SADT).* MS thesis, AFIT/GCS/ENG/88S-1, School of Engineering, Air Force Institute of Technology(AU), Wright-Patterson AFB, OH, September 1988.

11. Luger, George F. and Stubblefield, William A. *Artificial Intelligence and the Design of Expert Systems.* Benjamin and Cummings Publishing Company, Redwood City, CA, 1989.

12. Pressman, Roger S. *Software Engineering: A Practioner's Approach.* Second edition, New York, McGraw-Hill Book Company, 1989.

13. Giarratano, Joseph C. and Riley, Gary. *Expert Systems: Principles and Programming.* PWS-KENT Publishing Company, Boston, 1989.

14. Ross, Douglas T. "Structured Analysis(SA): A language for Communicating Ideas," *IEEE Transactions on Software Engineering*, SE-3, No.1:16-34, January 1977.

15. Ross, Douglas T. and K. E. Schoman "Structured Analysis for Requirements Definition," *IEEE Transactions on Software Engineering*, SE-3, No.1:6-15, January 1977.

16. Rowe Neil C. *Artificial Intelligence Through Prolog.* Prentice Hall, Englewood Cliffs, New Jersey, 1988.

17. SofTech, Inc. *Guide to Understanding Activity Diagrams* $IDEF_0$. Waltham, MA, September 1979(7500- 12).

18. Sommerville, Ian. *Software Engineering. third edition* Addison-Wesley Publishing Company, 1989.

19. Sun Micorsystems, Inc. *Pixrect Reference Manual.* Mountain View, CA, May 1988(800-1785-10).

20. Sun Microsystems, Inc. *SunView Programmer's Guide.* Mountain View, CA, September 1986(800-1345-10).

21. Yau, Stephen S. and Tsai, Jeffery J. "Knowledge Representation of Software Component Interconnection Information for Large-Scale Software Modifications," *IEEE Transactions on Software Engineering*,SE-13, No.3:355-361, March 1987.

22. Hartrum, Thomas C. *Readings II - Requirements Analysis.* Department of Electrical and Computer Engineering, Class notes, Draft # 2, Air Force Institute of Technology(AU), Wright-Patterson AFB, OH, January 1988.

23. Johnson, W. Lewis and Yue, Kaizhi "An Integrated Specfication Development Framework," *Information Sciences Institute*, University of Southern California, CA, October, 1988.

24. Moore Michael J. and Sheffield James Rodney "A PDL Synthesizer for Real-Time Systems," *SofTech, Inc.*, Dayton, OH, 1989.

25. Cureton, Bill "Program Synthesis: A Paradigm for Knowledge-Based Software Engineering," *Sun Technology*, 21-25, Winter, 1989.

## *Vita*

Intaek Kim [redacted] the son of BongYou and PungOk Kim. He graduated from the Ohyoun high school in 1978 and enterd the Korea Air Force Academy from which he received the major degree of a Bachelor of Engineering in Electrical Engineering in March 1983. He became a service man on active duty in March 1983, attended Seoul National University and received a Bachelor of Science Degree with major in Computer Science and Statistics in Feburary 1986. After graduation, He was assigned to R.O.K. Air Force Academy where served as Instructor in the Department of Computer Science and Statistics on CheungJu, Korea until March 1988. He entered the school of Engineering, Air Force Institute of Technology, Wright- Patterson Air Force Base, Ohio, in May 1988.

[redacted]

SECURITY CLASSIFICATION OF THIS PAGE

# REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION
Unclassified

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE

3. DISTRIBUTION / AVAILABILITY OF REPORT
Approved for public release
Distribution unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
AFIT/GCS/ENG/90J-02

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
School of Engineering

6b. OFFICE SYMBOL *(If applicable)*

7a. NAME OF MONITORING ORGANIZATION

6c. ADDRESS *(City, State, and ZIP Code)*
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433-6583

7b. ADDRESS *(City, State, and ZIP Code)*

8a. NAME OF FUNDING / SPONSORING ORGANIZATION
OSD/SDIO

8b. OFFICE SYMBOL *(If applicable)*
S/BM

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS *(City, State, and ZIP Code)*
Pentagon
Washington, DC 20301-7100

10. SOURCE OF FUNDING NUMBERS

| PROGRAM ELEMENT NO. | PROJECT NO. | TASK NO | WORK UNIT ACCESSION NO. |
|---|---|---|---|
| | | | |

11. TITLE *(Include Security Classification)*
EXPERT SYSTEM IN SOFTWARE ENGINEERING USING
STRUCTURED ANALYSIS AND DESIGN TECHNIQUE (SADT)

12. PERSONAL AUTHOR(S)
Intaek Kim, Captain Republic of Korea Air Force

13a. TYPE OF REPORT
Thesis

13b. TIME COVERED
FROM ________ TO ______

14. DATE OF REPORT *(Year, Month, Day)*
1990, June

15. PAGE COUNT
232

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

| FIELD | GROUP | SUB-GROUP |
|---|---|---|
| 12 | 05 | |
| | | |

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Software Engineering, Knowledge-Based System,
CASE tool,Predicate representation

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This thesis investigation presents the development of an application of expert system for checking the syntax of Structured Analysis and Design Technique (SADT) method. The tool provides the requirements analyst and the designer with a means of checking the syntax of IDEF0 diagram. The tool was implemented using expert system thechnique. The syntax checking process allows the extension of the rule base for the syntax knowledge representation of SADT methodology.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
□ UNCLASSIFIED/UNLIMITED □ SAME AS RPT. □ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL
Dr. Gary B. Lamont

22b. TELEPHONE *(Include Area Code)*
513-256-1279

22c. OFFICE SYMBOL
AFIT/ENG

Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE